Algoritmos para el problema de Flujo Máximo

Victoria Gracia Gracia

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Director del trabajo: Pedro Mateo Collazos
28 de Junio de 2019

Prólogo

El problema de obtención del máximo flujo que puede circular por una red de distribución es un problema importante del campo de las redes de flujo dentro de la Investigación Operativa.

Esta memoria se dedica a desarrollar un tipo de algoritmos de resolución que completan los resultados adquiridos previamente en el grado.

Se desarrollarán detalladamente los algoritmos junto con los resultados teóricos que lo sustentan.

También se incluirá un sencillo experimento computacional para comparar el comportamiento de un nuevo algoritmo con el comportamieno de otro algoritmo para este tipo de problemas visto en el grado.

Para el desarrollo de este trabajo se han usado tanto herramientas de la asignatura de Grafos y Combinatoria, como las de Investigación Operativa. También se ha manejado el lenguaje de programación Java, el cual no había utilizado durante el grado, y se ha utilizado el programa *R* aprendido en la asignatura Introducción a la Probabilidad y la Estadística.

Abstract

Flow network models are very important in the field of the Operation Research. These types of problems can model numerous problems and they also have very efficient resolution algorithms.

This Final Degree Project (FDP) is focused on the study of the Maximum Flow Problem (MFP). This problem is based on the following idea, given a product for which a distribution network is available, it is intended to find what is the maximum quantity of product that can be sent by that network.

It is common to find these kinds of problems in real life. For instance, these problems are used to assign tasks, matrix rounding, to optimize transport networks, electrical current, liquid or gas flow across connected pipes and tanker scheduling, just to name a few.

Making use of graph theory, the flow networks of the MFPs are represented by a directed graph, composed of a set of nodes and a set of arcs that join them.

The graph model associates an initial node as a source node and a final node as a sink node. Therefore, the MFP seeks to distribute the flow optimally from the source node to the sink node so that the maximum flow is found.

Throughout this work two types of procedures are shown, the Ford-Fulkerson algorithm, which was studied during the degree, and the preflow-push algorithms, which for a few years have become the most efficient techniques to solve MFPs.

Therefore, this proyect consists of three parts.

In the first chapter, the first section talks about some real examples that are modeled and solved as a problem of maximization of the flow through a network. Then, the basic concepts of graph theory are presented. As well, the necessary theory is introduced to present and manipulate the maximum flow problem. Later, the Ford-Fulkerson algorithm is presented, where these algorithms look for the maximum flow through paths from the source node to the sink node. Also the basic pseudocode of this algorithm will be seen and its computational complexity will be known. Next, a brief example is presented where we can see the iterations and the resulting networks. At the end of the chapter, variants and extensions for these problems are shown.

The second chapter is devoted to present the preflow-push algorithms. These types of algorithms send flow saturating arcs, causing the appearance of nodes with excess flow. It begins by presenting the fundamental theory and tools for the development of the algotihm (labels, distances,...). Subsequently, the algorithm will be presented, where its pseudocode can also be seen. Later, the same example solved in the first chapter is solved with this algorithm. Moreover, its computational complexity will be discussed. In the last section the correction of the algorithm will be studied and three possible modifications will be named along with their computational complexity. Finally, we explain briefly how one of these methods works, the preflow-push algorithm FIFO.

In the third chapter, an implementation in Java language of the algorithms has been carried out and compared with one Ford-Fulkerson implementation. We have built and solved several synthetic sets of MFP, the problems present variants in the maximum arc capacity, the total supply and the number of arcs. With the results, a file has been created and then studied in *R* software to obtain some summary measures of the obtained data. You can also see graphs where the average times of these problems are represented. Finally, I present some comments of the results and conclusions.

Índice general

Capítulo 1

Problema de Flujo Máximo y algoritmo de Ford y Fulkerson

Los modelos de redes de flujo y en particular el problema de flujo máximo son muy importantes en el campo de la Investigación Operativa. Mediante estos se pueden modelar numerosos e importantes problemas, además los algoritmos de resolución suelen ser muy eficientes comparados con las de otras disciplinas. Los modelos de redes tienen muy buena acogida por parte de los no especialistas en el campo debido a su aspecto visual. En la literatura pueden encontrarse numerosos problemas relacionados con redes: ruta mínima, máximo flujo, flujos con costo mínimo, asignación/matching, viajante, transporte, coloreado de grafos, etc.

Este trabajo se centra en el problema de máximo flujo (PMF). La idea general es dado un sistema de distribución de un cierto item para el cual se dispone de una red de distribución, ha de determinarse la máxima cantidad de producto que puede enviarse simultáneamente por dicha red. Por ejemplo, dado un sistema de alcantarillado de una ciudad encontrar cual sería la capacidad máxima de evacuación de agua proporcionado por dicho sistema.

Este problema fue tratado en la asignatura Grafos y Combinatoria de primer curso del grado, donde se introdujo y se estudió el algoritmo clásico de optimización presentado por los creadores del problema en el año 1956, el algoritmo de Ford y Fulkerson [4]. Este fue el primer algoritmo propuesto para resolver el problema de flujo máximo.

En este proyecto se propone estudiar otro tipo de algoritmos de resolución que funcionan de una forma diferente al previamente estudiado. En lugar de basar el funcionamiento en buscar rutas en la red de distribución por las que se puede ir incorporando items del producto, se trata de provocar atascos en los nodos de la red enviando todo el flujo que se puede hacia ellos y posteriormente redistribuyendo, empujando, esos excesos que se han formado en los nodos.

En las siguientes secciones se verán algunas de las aplicaciones más frecuentes de estos problemas, así como se introducirán los conceptos básicos y la teoría necesaria para plantear, posteriormente, el problema de flujo máximo.

1.1. Aplicaciones del problema de Flujo Máximo

El problema de máximo flujo y el denominado problema del mínimo corte[1] surgen en una amplia variedad de situaciones y en varias formas. Por ejemplo, pueden aparecer como subproblemas de otros problemas más complejos, como en el problema de flujo de mínimo coste o el de flujo generalizado. En otras ocasiones estos problemas surgen de una forma natural: en la asignación de módulos informáticos a los procesadores de los ordenadores, en redondeo de datos del censo para conservar la confidencialidad de los hogares individuales, en programación de camiones cisterna, para optimizar redes de agua, gas,

[1] El problema de flujo máximo tiene asociado un problema equivalente denominado problema del mínimo corte, ambos se relacionan por el hecho de que el valor del máximo flujo que puede circular por una red de flujo coincide con la capacidad mínima de todos los cortes que se pueden hacer en la red.

eléctricas, para circulación con demandas, programación de vuelos,... Estas aplicaciones y algunas otras pueden consultarse en [11] y [1].

A continuación, se comentan algunas de las aplicaciones más interesantes, no tan directas como las derivadas de redes de distribución.

1.1.1. Problema de flujo factible

En este problema se tiene un conjunto de elementos de la red, los cuales ofertan cada uno de ellos una cierta cantidad fija de un item, y otro conjunto de elementos que demandan otra cierta cantidad de dicho item.

Entre los ofertantes y los demandantes existe una red de conexiones con distintas capacidades de paso. La cuestión es determinar si existe un envío factible, es decir, si es posible que los demandantes reciban las cantidades solicitadas.

Véase con un ejemplo, se supone que una mercancía está disponible en algunos puertos marítimos y esta es deseada por otros puertos. Se conoce el stock de mercancía disponible en los puertos, la cantidad requerida por los otros puertos y la máxima cantidad de mercancía que se puede enviar en esa ruta marítima. Por tanto, lo que se pretende es saber si se pueden satisfacer todas las demandas utilizando los suministros disponibles.

El modelado como un problema de máximo flujo se explica en el Apéndice A, ya que todavía no se han definido los elementos de redes necesarios para su formulación.

1.1.2. Problema de los representantes

Este tipo de problemas tienen varias aplicaciones en configuraciones de asignación de recursos. Lo que se pretende es encontrar una situación de equilibrio en la que dado un número de representantes, a cada uno de estos se les asocie una determinada actividad o trabajo. Véase a continuación con un ejemplo.

Dada una ciudad donde hay un determinado número de ciudadanos que viven ahí, un conjunto de asociaciones vecinales y otro conjunto de partidos políticos. Cada uno de los ciudadanos pertenece al menos a una de esas asociaciones vecinales y solo puede pertenecer exactamente a un partido político. Cada asociación vecinal debe de asignar a uno de sus miembros para que este lo represente en el consejo de gobierno de la ciudad, así se tendrá un máximo de miembros que pertenezcan a cada partido político.

Cabe destacar que una vez encontrado el flujo máximo de esta red, en caso de que este valor de flujo máximo fuese igual al número de asociaciones vecinales que se dispone, se tendría un consejo equilibrado, ya que así cada asociación vecinal tendría representación. En caso contrario, no estaría equilibrado.

En el Apéndice A puede consultarse como se resolvería este problema como PMF.

1.1.3. Problema de redondeo de matrices

Este tipo de problemas se centran en el redondeo de los elementos de una matriz, suma de filas y suma de columnas.

Dada una matriz de números reales en la que se pueden realizar sumas de filas y de columnas. Se puede redondear cualquier número real al siguiente entero más pequeño o al siguiente entero más grande. El problema de redondeo de una matriz requiere que se redondeen los elementos de la matriz, las sumas de las columnas y las filas, de modo que la suma de los elementos redondeados en cada fila sea igual a la fila redondeada, e igualmente en el caso de las columnas. Es decir, se realiza un redondeo consistente.

Nótese que se puede intentar descubrir el esquema de redondeo, en caso de que exista, resolviendo un problema de flujo factible para una red con límites inferiores no negativos en los flujos de arco.

Esta técnica se usa en la *Oficina del Censo de los Estados Unidos*, donde se utiliza la información del censo de población para construir millones de tablas en una amplia variedad de propósitos. Legal-

mente, debe de protegerse el origen de la información y no divulgar estadísticas que pueden asociarse a cualquier individuo, por lo que se trata de enmascarar la información.

Se trata de modificar los datos de forma que proporcionen los mismos valores estadísticos pero que no permita determinar de que individuo puede provenir la información.

En el Apéndice A puede consultarse con un pequeño ejemplo como se convertiría un problema de este tipo a un PMF.

Además en estos problemas, se podría realizar una correspondencia uno a uno entre los redondeos consistentes de la matriz y los flujos factibles de su red. En consecuencia, se puede encontrar un redondeo consistente resolviendo un problema de flujo máximo en la red correspondiente.

Las aplicaciones citadas anteriormente, entre otras, se pueden consultar más detalladamente en [1].

1.2. Notaciones y conceptos básicos sobre Teoría de Grafos y Redes de Flujos

En esta sección se presenta la notación y elementos relacionados con los grafos y redes de flujo para, posteriormente, plantear el problema de flujo máximo.

A continuación se describen los elementos principales que definen una red.

- Un grafo $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ está compuesto por un conjunto de elementos denominados *vértices o nodos*, denotado por $\mathscr{N}$, donde

$$\mathscr{N} = \{1, 2, \ldots, n\},\ |\mathscr{N}| = n, \tag{1.1}$$

  y un conjunto de pares, denominados *arcos o aristas*, contenidos en el producto cartesiano de $\mathscr{N}$ y denotado por $\mathscr{A}$,

$$\mathscr{A} = \{(i, j) \mid i, j \in \mathscr{N}\} \subset \mathscr{N} \times \mathscr{N},\ |\mathscr{A}| = m. \tag{1.2}$$

  Nótese que los arcos o aristas representan un enlace entre un par de nodos.

- Los arcos o aristas pueden ser de dos tipos dependiendo de si el orden de los nodos es relevante o no. Un arco se dirá *dirigido* si el orden de los nodos es relevante y en cuyo caso el arco (i, j) será distinto del arco (j, i), y se dirá *no dirigido* si el orden no importa, en cuyo caso el arco (i, j) será igual al arco (j, i).

- Dado un arco (i, j), sus nodos i con j se llamarán *extremos del arco*. Además, los nodos i y j se dirá que son *adyacentes o vecinos*.

- *Un grafo no dirigido*, es un grafo $\mathscr{G} = (\mathscr{N}, \mathscr{A})$, donde todos sus arcos son no dirigidos, mientras que un grafo en el que sus arcos sean dirigidos se denominará *grafo dirigido*. Para el problema que se va a tratar, problema de flujo máximo, se considerarán redes dirigidas. En caso de existir algún arco no dirigido se sustituirá por los correspondientes dos arcos dirigidos que lo representen. Es decir, dado (i, j) no dirigido se reemplazará por los dos arcos dirigidos (i, j) y (j, i).

- Cuando sea necesario en el grafo se destacarán dos nodos con propiedades especiales, el *nodo fuente u origen*, el cual denotaremos como nodo 1 y es el nodo de entrada a la red, y el *nodo destino o final*, denotado por n y el cual será el nodo de salida de la red.

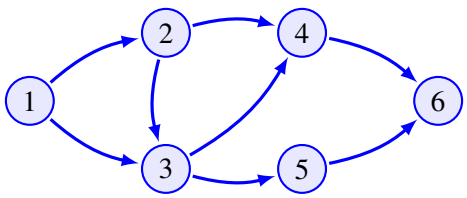

Figura 1.1: Dibujo representativo de un grafo de seis nodos y ocho arcos dirigidos.

Definición 1. Dado un grafo dirigido $\mathscr{G} = (\mathscr{N}, \mathscr{A})$, con $\mathscr{N} = \{1, 2, \ldots, n\}$ y $\mathscr{A} = \{(i, j) \mid i, j \in \mathscr{N}\}$ se definen:

- $A(i) = \{j \in \mathscr{N} \mid (j, i) \in \mathscr{A}\}$, *conjunto de antecesores de un nodo* $i \in \mathscr{N}$.
- $D(i) = \{j \in \mathscr{N} \mid (i, j) \in \mathscr{A}\}$, *conjunto de sucesores de un nodo* $i \in \mathscr{N}$.

Definición 2. Dado $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ una *trayectoria* T desde un nodo i_1 a un nodo i_s es un conjunto $i_1, i_2, \ldots, i_{s-1}, i_s$ de nodos distintos y una sucesión de $s-1$ arcos tal que el $j-ésimo$ arco de la trayectoria es de la forma (i_j, i_{j+1}) o (i_{j+1}, i_j). En el primer caso (i_j, i_{j+1}) es un arco *hacia adelante* de la trayectoria y en el segundo (i_{j+1}, i_j) es un arco *hacia atrás*.

Se denotará mediante T^+ al conjunto de arcos hacia adelante de una trayectoria T y T^- al conjunto de arcos hacia atrás.

Definición 3. Dada una trayectoria T desde un nodo i_1 a un nodo i_s, si todos sus arcos son de la forma (i_j, i_{j+1}) entonces dicha trayectoria se denomina *cadena* y se denotará como C.

Definición 4. Dado un grafo $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ se dice *simple* si para todo $i, j \in \mathscr{N}$ existe un único arco $(i, j) \in \mathscr{A}$ que los une. Es decir, un grafo simple es un grafo en el que existe un solo arco que une dos nodos distintos.

Definición 5. Se dice que un grafo $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ es *conexo* si para cualquier par de nodos de dicho grafo existe una trayectoria entre ellos. Es decir, para cualesquiera nodos i, j existe una posible trayectoria de i a j.

Notación. Para simplificar notación dados $X, Y \subseteq \mathscr{N}$, se denota $(X, Y) = \{(i, j) \in \mathscr{A} | i \in X, j \in Y\}$.

Dadas funciones $g : \mathscr{N} \to \mathbb{R}$ que asigna g_i, $\forall\, i \in \mathscr{N}$ y $h : \mathscr{A} \to \mathbb{R}$ que asigna $h_{ij}\, \forall\, (i, j) \in \mathscr{A}$ y dados $X, Y \subset \mathscr{N}$ se denota:

$$g(X) = \sum_{i \in X} g_i \quad \text{y} \quad h(X, Y) = \sum_{(i,j) \in (X,Y)} h_{ij}.$$

Estos conceptos básicos y definiciones pueden consultarse con más detalle en [1], [6] y [5].

Dada una red por la que fluye un material desde el origen, lugar donde se produce a ritmo constante, hasta el destino, donde es consumido a ese mismo ritmo. Cada arco de la red representa un conducto por el que dicho material circula.

Además, puede suponerse que cada arco tiene asociado un límite de capacidad, el cual representa la máxima cantidad de material que puede circular por él.

En el caso de los nodos distintos del nodo origen y nodo destino, estos se consideran puntos de interconexión, en los cuales ha de cumplirse que la cantidad de material que llega a esos nodos debe de salir de ellos al mismo ritmo al que entra, es decir, sin producir pérdidas. Esta propiedad se conoce como "Ley de conservación del flujo" o también como la conocida Ley de conservación de corrientes de Kirchoff [6].

A continuación se ven algunas definiciones importantes para abordar los problemas de flujo máximo.

Definición 6. Una red de flujo es un grafo simple y conexo $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ donde:

- Hay dos nodos distinguidos, el nodo origen 1 y el nodo destino n.
- Se define una función $f : \mathscr{A} \to \mathbb{R}^+$ tal que para cada arco $(i, j) \in \mathscr{A}$ le asigna la cantidad de flujo, $f(i, j) = f_{ij}$ de un cierto item que se envía desde el nodo i al nodo j.
- Se definen también $l : \mathscr{A} \to \mathbb{R}^+$ y $u : \mathscr{A} \to \mathbb{R}^+$ funciones capacidad inferior y superior del arco, que representa la mínima y máxima cantidad de flujo que pueden circular por un arco $(i, j) \in \mathscr{A}$, $l(i, j) = l_{ij}$ y $u(i, j) = u_{ij}$.

En lo que sigue se asume también, sin pérdida de generalidad, que si existe el arco $(i,j) \in \mathscr{A}$ entonces también existirá el arco (j,i), esto no supone ningún problema ya que si la red original no lo contiene se incorpora a esta extendiendo las funciones l,u de tal forma que $l_{ji} = u_{ji} = 0$ [2].

Definición 7. Un *flujo factible* sobre una red de flujo $\mathscr{G} = (\mathscr{N},\mathscr{A})$ es un flujo $f : \mathscr{A} \to \mathbb{R}^+$ que satisface las siguientes restricciones:

$$\sum_{j \in D(i)} f_{ij} - \sum_{j \in A(i)} f_{ji} = \begin{cases} v & \text{si } i = 1 \\ 0 & \text{si } i \in \mathscr{N} \setminus \{1,n\} \\ \text{-}v & \text{si } i = n \end{cases} \tag{1.3}$$

$$0 \leq l_{ij} \leq f_{ij} \leq u_{ij} \; \forall (i,j) \in \mathscr{A}. \tag{1.4}$$

El primer conjunto de ecuaciones (1.3) se conocen como *restricciones de conservación del flujo* y el segundo (1.4) se nombra como *restricción de cotas*.

A la cantidad v se le denomina *valor del flujo* y representa el flujo neto que atraviesa la red, entrando por el nodo 1 y saliendo por el nodo n,

$$v = \sum_{j \in D(1)} f_{1j} - \sum_{j \in A(1)} f_{j1} = \sum_{j \in A(n)} f_{jn} - \sum_{j \in D(n)} f_{nj}.$$

Definición 8. Dado un flujo factible sobre una red $\mathscr{G} = (\mathscr{N},\mathscr{A})$ de valor v y una trayectoria T de 1 a n, se dice que T es una *trayectoria de aumento de flujo* si $f_{ij} < u_{ij} \; \forall (i,j) \in T^+$ y $f_{ij} > l_{ij} \; \forall (i,j) \in T^-$.

Definición 9. Dado un flujo factible sobre una red $\mathscr{G} = (\mathscr{N},\mathscr{A})$, se llama *capacidad residual* del arco (i,j) a $r_{ij} = u_{ij} - f_{ij} + f_{ji} - l_{ji}$. Dicha cantidad se corresponde con el flujo adicional que se puede enviar desde el nodo i al nodo j usando los arcos (i,j) y (j,i).

Definición 10. Dado un flujo factible sobre una red $\mathscr{G} = (\mathscr{N},\mathscr{A})$ se denomina *red residual* a $\mathscr{G}_R = (\mathscr{N},\mathscr{A}_R)$ donde $\mathscr{A}_R = \{(i,j) \in \mathscr{A} \mid r_{ij} > 0\}$.

Obsérvese que una trayectoria de aumento de flujo sobre la red $\mathscr{G} = (\mathscr{N},\mathscr{A})$ corresponde a una cadena en la red $\mathscr{G}_R = (\mathscr{N},\mathscr{A}_R)$.

Definición 11. Dado un flujo f, los arcos (i,j) que cumplen $f_{ij} = u_{ij}$ se dice que están *saturados*, y los que cumplen $f_{ij} = l_{ij}$ son *nulos*.

De acuerdo a las definiciones 2 y 8, dado un flujo definido sobre $\mathscr{G} = (\mathscr{N},\mathscr{A})$ y una trayectoria de aumento, entonces todos los arcos hacia adelante son *no saturados* y los arcos hacia atrás son *no nulos*.

Definición 12. Dada una red de flujo, un *corte* separando 1 de n, es un conjunto de arcos $(X,\bar{X})$ donde $1 \in X$ y $n \in \bar{X}$.

Definición 13. Dada una red de flujo $\mathscr{G} = (\mathscr{N},\mathscr{A})$, y un corte $(X,\bar{X})$ separando los nodos 1 y n. Entonces la cantidad

$$c(X,\bar{X}) = u(X,\bar{X}) - l(\bar{X},X), \tag{1.5}$$

se llama *capacidad del corte* $(X,\bar{X})$.

Es decir, al realizar un corte la red de flujo se divide en dos redes, por lo que si se eliminasen los arcos correspondientes a dicho corte no podría enviarse el material desde 1 a n. Además, de forma intuitiva, parece claro que no se puede enviar más flujo que el que "cabe" por esos arcos del corte. El siguiente lema formaliza esa idea.

Lema 1.1. *Sea $\mathscr{G} = (\mathscr{N},\mathscr{A})$ una red de flujo, f un flujo sobre esa red y $(X,\bar{X})$ un corte cualquiera separando 1 y n. Entonces,*

$$v = f(X,\bar{X}) - f(\bar{X},X) \leq c(X,\bar{X}) = u(X,\bar{X}) - l(\bar{X},X) \text{ [3]}. \tag{1.6}$$

[2] Se realiza para simplificar los desarrollos teóricos.

[3] Las demostraciones de la mayoría de los resultados de este capítulo se omiten ya que se estudiaron en su momento en la asignatura de Grafos y Combinatoria.

Obsérvese que si se encuentra un flujo f y un corte $(X,\bar{X})$ separando 1 y n que cumpla $v = c(X,\bar{X})$, necesariamente ese flujo es máximo y ese corte tiene capacidad mínima, además debe ser $f(X,\bar{X}) = u(X,\bar{X})$ y $f(\bar{X},X) = l(\bar{X},X)$. Es decir, por los arcos de $(X,\bar{X})$ circula un flujo igual a su capacidad máxima, y por los arcos de $(\bar{X},X)$ circula el mínimo flujo factible.

Estas ideas permiten enunciar el siguiente teorema, resultado fundamental de esta teoría.

Teorema 1.2. ***Teorema de máximo flujo mínimo corte***
Dada una red de flujo $\mathscr{G} = (\mathscr{N},\mathscr{A})$. El valor máximo de v de los flujos factibles de la red coincide con la mínima capacidad de todos los cortes que separan los nodos 1 y n de la red, es decir,

$$\max_{f \text{ flujo factible en } \mathscr{G}=(\mathscr{N},\mathscr{A})} v = \min_{(X,\bar{X}) \text{separando 1 y } n} c(X,\bar{X}). \tag{1.7}$$

Todos los resultados anteriores se pueden consultar con detalle en [1], [6], [5] y [3].

1.3. Problema de Flujo Máximo y Algoritmo de Ford y Fulkerson

1.3.1. Problema de Flujo Máximo

Dada una red de flujo, el problema de flujo máximo consiste en determinar un flujo factible con el máximo valor v posible, es decir[4]:

$$\begin{aligned}
&\max v \\
&\text{sujeto a :} \\
&\sum_{j\in D(i)} f_{ij} - \sum_{j\in A(i)} f_{ji} = \begin{cases} v & \text{si } i = 1 \\ 0 & \text{si } i \in \mathscr{N}\setminus\{1,n\} \\ -v & \text{si } i = n \end{cases} \\
&0 \le l_{ij} \le f_{ij} \le u_{ij} \quad \forall (i,j) \in \mathscr{A}.
\end{aligned} \tag{1.8}$$

A continuación se mostrará el algoritmo de resolución propuesto por Ford y Fulkerson en el año 1956 [5]. Para ello se enuncia el siguiente lema, donde se basa el funcionamiento del algoritmo.

Lema 1.3. *Sea f un flujo sobre la red $\mathscr{G} = (\mathscr{N},\mathscr{A})$. Si existe una trayectoria de aumento para f, entonces existe otro flujo f' sobre $\mathscr{G} = (\mathscr{N},\mathscr{A})$ tal que $v' > v$. Equivalentemente, si no existen dichas trayectorias de aumento, entonces f es el flujo máximo.*

1.3.2. Algoritmo de Ford y Fulkerson

El procedimiento de Ford y Fulkerson para calcular un flujo máximo es el siguiente [6].

1. (Paso inicial) Comenzar con un flujo factible inicial cualquiera f^0, en lo que asumiremos que las cotas inferiores l_{ij} son cero, de esta forma el desarrollo es más sencillo y no se pierde generalidad, un problema con cota inferior factible puede transformarse en un problema sin cotas sin más que calcular un flujo factible inicial, por ejemplo, con el flujo nulo $f^0_{ij} = 0$, $\forall (i,j) \in \mathscr{A}$, $t = 0$.

2. (Paso genérico) Buscar una trayectoria de aumento para f^t. Si existe una trayectoria de aumento para f^t, se obtiene otro flujo f^{t+1} tal que $v^{t+1} > v^t$. Sin más que sumar o restar al flujo actual el flujo de los arcos de las trayectorias (sumas en T^+ y restas en T^-). Se considera $t = t+1$ y se repite el paso genérico. Si no existe un camino de aumento se finaliza. El último flujo obtenido es el óptimo.

[4] Obsérvese que considerando f_{ij} como variable se estaría ante un problema de programación lineal.

Ford y Fulkerson demuestran que bajo la suposición de que las cotas del flujo son enteras[5]entonces el algoritmo converge y mantiene valores de f_{ij} enteros $\forall (i,j) \in \mathscr{A}$.

Teorema 1.4. *Sea $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ una red de flujo con capacidades superiores e inferiores del flujo de los arcos, u_{ij} y l_{ij}, enteras. Entonces, comenzando con un flujo factible entero, el proceso anterior siempre termina y además los valores $f_{ij}^t \forall (i,j) \in \mathscr{A}$ en todo el proceso son siempre enteros.*

Es decir, a partir de un flujo cualquiera f^0 , mediante el método de las trayectorias de aumento, se van obteniendo flujos con mayor valor estricto. Cuando este proceso termina, el flujo obtenido es el máximo. De hecho, se demuestra que al finalizar se puede construir un corte $(X, \bar{X})$ para el cual su capacidad coincide con el valor de flujo y por el Teorema 1.2 el flujo es máximo.

Un esquema básico del pseudocódigo del algoritmo de Ford y Fulkerson es el siguiente.

Algorithm 1: Algoritmo básico de Ford y Fulkerson

```
Data: G, 1, n, f, v
Result: flujo óptimo
foreach arco (i,j) ∈ 𝒜 do
    f_ij = 0
end
begin
    pred(j) = 0 para cada j ∈ 𝒩;
    etiquetar nodo 1, Lista= {1}, pred(1) = −1;
    while Lista≠ ∅ o n no tiene etiqueta do
        Extraer i de Lista;
        foreach (i,j) ∈ 𝒜_R : j ∈ D(i) do
            if j no etiquetado then
                pred(j) = i, etiqueta el nodo j, Lista = Lista ∪ {j};
                if j = n then
                    ir a ∗ ;
                end
            end
        end
        ∗ if n etiquetado then
            j=n;
            T = {∅};
            while j ≠ −1 do
                T = T ∪ {(pred(j), j)};
                j = pred(j);
            end
            v = min{r_ij | (i,j) ∈ T};
            foreach cada arco (i,j) ∈ T do
                if (i,j) ∈ 𝒜 then
                    f_ij = f_ij + v;
                else
                    f_ji = f_ji − v;
                end
            end
        end
    end
end
```

[5]En el caso de cotas no enteras no se puede garantizar la convergencia del algoritmo, de hecho se encuentran en la literatura sencillos ejemplos de redes con capacidades superiores racionales e inferiores nulas para los que el algoritmo no finaliza [1].

Nótese que este algoritmo corre en un tiempo pseudopolinomial. Sea U el número que representa la capacidad que delimita los arcos, estos son enteros y finitos, y dado n que representa el número de nodos, se tiene que la capacidad del corte mínimo es a lo sumo nU. En consecuencia, el valor del flujo máximo estará limitado por nU y como el algoritmo aumenta el valor de flujo en al menos una unidad en cualquier momento, este finalizará en nU aumentos. Cada uno de estos aumentos requiere un tiempo $O(m)$ debido a que el algoritmo examina a lo sumo todos los de la red una vez, luego la complejidad es $O(m)$ veces el número de aumentos. Como hay nU aumentos, en consecuencia, el límite de ejecución del algoritmo es $O(nmU)$.

A continuación se muestra un pequeño ejemplo para observar el procedimiento del algoritmo de Ford y Fulkerson.

Ejemplo 1. Se parte de la siguiente red, en la cual pueden observarse las capacidades sobre cada arco. Cabe destacar que inicialmente todos los arcos poseen flujo cero, por lo que las capacidades residuales coinciden con la capacidad del arco.

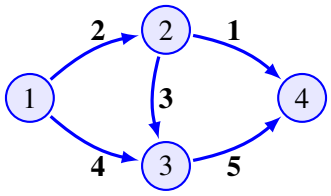

Figura 1.2: Grafo inicial del problema.

- En la primera iteración se busca una trayectoria de aumento, tras un etiquetado (como el que puede consultarse en el pseudocódigo) se obtiene la trayectoria que pasa por los nodos 1, 3 y 4. Su capacidad será $min\{4,5\} = 4$, luego el flujo máximo en esta iteración es 4. Ahora el flujo de la red cambia, el arco que une los nodos 1 y 3 queda saturado, el que une 3 y 4 tiene como capacidad 1, por lo que el de 4 a 3 tiene capacidad de 4 (Figura 1.3 izquierda).

- En la segunda iteración, se obtiene la trayectoria de aumento que une los nodos 1, 2 y 4. En este caso la capacidad es $min\{2,1\} = 1$. Ahora el flujo máximo será $4+1 = 5$. Luego, se vuelven a actualizar los flujos de los arcos de la red (Figura 1.3 centro).

- En la tercera iteración se selecciona la única trayectoria de aumento posible, esta es la que pasa por los nodos 1, 2, 3 y 4. La capacidad es $min\{1,3,1\} = 1$. Con lo que tras enviar ficho flujo se obtiene como valor 6, que es un flujo máximo. Por lo que el flujo resultante sobre los arcos quedará como puede observarse en la Figura 1.3 derecha.

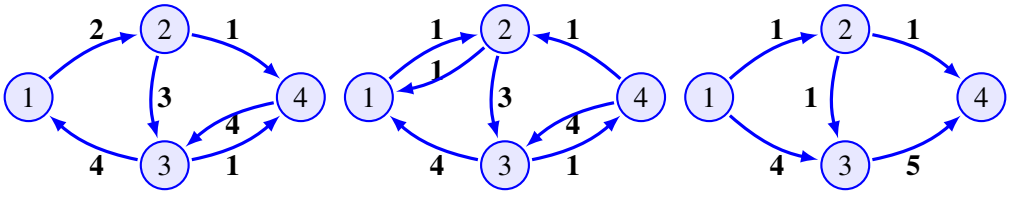

Figura 1.3: Las dos primeras redes representan las redes residuales tras la primera y segunda iteración, respectivamente. Y la figura de la derecha representa la red de flujo resultante.

No se pueden tomar más caminos de aumento, por lo tanto el algoritmo finaliza. Así pues, se tiene que el flujo máximo es 6. Además, un corte de capacidad mínima asociado a dicho flujo será: $X = \{1\}$, $\bar{X} = \{2,3,4\}$, así $(X,\bar{X}) = \{(1,2),(1,3)\}$ y $c(X,\bar{X}) = 6$, que coincide con el flujo máximo.

1.4. Variantes y extensiones del problema reducibles a un problema de Flujo Máximo

Se pueden encontrar problemas de flujo máximo con diferentes variantes, a continuación se verán algunas de las más comunes [6].

- Uno de los casos más frecuentes en la práctica es encontrar redes en las cuales existen varios nodos origen. En estos se puede producir el material a enviar con unos determinados límites sobre la cantidad de material producida, y además, en estas redes también pueden existir varios nodos destino donde el material será consumido con unos límites sobre el máximo ritmo al que dicho material puede ser consumido. Para abordar este tipo de problemas se añaden a la red dos nuevos nodos, un super-nodo origen, el cual se conecta mediante arcos con cada uno de los nodos origen y se les asigna una capacidad determinada. Así mismo, se crea un super-nodo destino que une todos los nodos destino con este, e igualmente se les asigna una capacidad. Considerando esta nueva red es posible resolver el problema como problema de flujo máximo y así obtener el flujo máximo del problema original.

  Este caso se corresponde con el primer ejemplo (1.1.1) de la primera sección, aplicaciones para el PMF.

- En algunos grafos pueden aparecer arcos no dirigidos, es decir, en los que el material puede circular por el mismo arco en un sentido u otro teniendo una capacidad asignada. Sustituyendo estos arcos por dos arcos, uno en cada sentido de circulación con la misma capacidad asignada, o diferente si la cantidad de producto que puede circular en cada sentido tiene límites distintos, se puede obtener la solución para este tipo de problemas.

- En algunas ocasiones pueden aparecer redes que poseen restricciones sobre la cantidad de flujo que puede circular por cada nodo, es decir, tienen un límite acerca del flujo total que puede entrar y/o salir en cada nodo. Para resolver este tipo de problemas se construye un nuevo grafo en el que cada nodo con restricciones de capacidad se sustituye por dos nodos (denotándose generalmente con los superíndices $+$ y $-$) que se conectan mediante un arco con una capacidad determinada. Una vez encontrado el máximo flujo para este nuevo grafo se obtiene el máximo flujo para el problema original.

Capítulo 2

Algoritmo de preflujo

Existen otro tipo de algoritmos más modernos, conocidos como algoritmos de empuje de flujo, que desde hace unos años se han convertido en técnicas más eficientes para resolver los problemas de máximo flujo, tanto teóricamente como computacionalmente. Estos algoritmos fueron desarrollados por Andrew V. Goldberg y Robert Tarjan en su trabajo, *A new approach to the maximum-flow problem*, [8], como una alternativa desde otro punto de vista a la resolución de los PMF.

Los algoritmos de preflujo no envían flujos a lo largo de trayectorias desde el nodo origen hasta el nodo destino, en su lugar, envían flujo saturando arcos y provocando la aparición de nodos con exceso, entonces dichos "atascos" se envían hacia el nodo n mediante los arcos que están más cerca del destino. Por lo que esta estrategia permite que estos algoritmos obtengan mayor aceleración que el resto de algoritmos para trayectorias de aumento.

Inicialmente, se comienza con un flujo que se va convirtiendo en un flujo máximo entre los nodos vecinos mediante la operación de empuje (push), y posteriormente realiza la operación de reetiquetado (relabel) para reetiquetar los flujos.

2.1. Propiedades y herramientas necesarias para los algoritmos de preflujo

Las etiquetas de distancia juegan un papel fundamental para implementar este tipo de algoritmos, como se verá a continuación.

Definición 14. Una *función de distancia* $d : \mathscr{N} \to \mathbb{Z}^+ \cup \{0\}$ con respecto a las capacidades residuales r_{ij} es una función del conjunto de nodos al conjunto de los enteros no negativos. Se dice que una función de distancia es *válida* con respecto a un flujo f si cumple las siguientes condiciones:

$$d(n) = 0; \tag{2.1}$$

$$d(i) \leq d(j) + 1, \text{ para cada arco } (i,j) \in \mathscr{A}_R. \tag{2.2}$$

Se hace referencia a $d(i)$ como la etiqueta de distancia del nodo i, y a las condiciones (2.1) y (2.2) como *condiciones de validez*.

Los siguientes resultados muestran porqué las etiquetas de distancia son útiles para diseñar algoritmos de distribución de flujo.

Propiedad 1. Si las etiquetas de distancia son válidas, la etiqueta de distancia $d(i)$ es una cota inferior en la longitud de la cadena más corta (dirigida) desde el nodo i al nodo n en la red residual.

Para validar esta propiedad, sea $i = i_1 - i_2 - \ldots - i_k - i_{k+1} = n$ y una cadena cualquiera de longitud k desde el nodo i al nodo n en la red residual. Las condiciones de validez, (2.1) y (2.2), implican:

$$
\begin{aligned}
&d(i_k) \leq d(i_{k+1}) + 1 = d(n) + 1 = 1,\\
&d(i_{k-1}) \leq d(i_k) + 1 \leq 2,\\
&d(i_{k-2}) \leq d(i_{k-1}) + 1 \leq 3,\\
&\qquad\qquad \vdots\\
&d(i) = d(i_1) \leq d(i_2) + 1 \leq k.
\end{aligned}
$$

Por tanto, para cualquier cadena de longitud k de i a n, $d(i) \leq k \,\forall\, k$ y $d(i)$ es una cota inferior de cualquier cadena incluida la de longitud mínima.

Lema 2.1. *Si $d(1) \geq n$, la red residual $\mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$ no contiene cadenas (trayectorias dirigidas) desde el nodo origen al nodo destino. Es decir, no existen trayectorias de aumento.*

Demostración. Por la Propiedad 1 se tiene que $d(i)$ es una cota inferior de la longitud de la cadena más corta de i a n. Suponer que $d(1)$ es una cota inferior y existe una cadena de 1 a n en la red residual, entonces la longitud sera menor o igual que $n-1$, pero por hipótesis $d(1) \geq n$, por tanto se tiene contradicción. Esto implica que no existe una cadena de 1 a n en $\mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$. □

Definición 15. Se dice que una etiqueta de distancia es *exacta* si, para cada nodo i, se tiene que $d(i)$ es igual a la longitud de la cadena más corta desde el nodo i al nodo destino, n, en $\mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$.

Definición 16. Se dice que un arco (i, j) en una red residual $\mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$ es *admisible* si cumple que $d(i) = d(j) + 1$. Y en caso contrario, se dirá que es *inadmisible*.

Así mismo, también se hace referencia a una cadena de 1 a n compuesta por arcos admisibles como *cadena admisible*.

Lema 2.2. *Una cadena admisible es la cadena más corta desde el nodo origen hasta el nodo destino sobre la red residual $\mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$.*

Demostración. Dado que cada arco (i, j) en una cadena admisible P es admisible, la capacidad residual de estos arcos y las etiquetas de distancia de sus nodos finales satisfacen:

1. $r_{ij} > 0$.
2. $d(i) = d(j) + 1$.

Por un lado, $r_{ij} > 0$, es decir, $u_{ij} - f_{ij} + f_{ji} - l_{ji} > 0$ y la Definición 8 implica que P es una cadena de aumento. Por otro lado, la segunda condición implica que si P contiene k arcos, entonces $d(1) = k$, ya que $d(n) = 0$ y se da la igualdad en la Propiedad 1.

Por esta Propiedad, sea $d(1)$ una cota inferior de la longitud de cualquier cadena desde el origen al destino en $\mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$, luego cualquier cadena será mayor o igual que $d(1)$ (porque es una cota inferior). Como se tiene una cadena de valor k, si esta alcanza exactamente la cota inferior, es la más pequeña de todas. Por tanto, se tiene que P debe ser la cadena de aumento más corta. □

Puede consultarse más información sobre las etiquetas de distancia en [1].

2.2. Definiciones, características y presentación del algoritmo

Los algoritmos de preflujo empujan los flujos por los arcos individuales en lugar de aumentar por medio de trayectorias, como consecuencia estos algoritmos no satisfacen las restricciones de conservación del flujo, (1.3), en etapas intermedias. De hecho, permiten que el flujo que entra a un nodo supere el flujo que sale de dicho nodo.

Se hace referencia a cualquier solución de este tipo como preflujo.

Definición 17. Un *preflujo* es una función $p : \mathscr{A} \longrightarrow \mathbb{R}$ que satisface la restricción de cotas, (1.4), y la siguiente restricción:

$$\sum_{j \in A(i)} f_{ji} - \sum_{j \in D(i)} f_{ij} \geq 0 \ \ \forall i \in \mathscr{N} \setminus \{1, n\}. \tag{2.3}$$

Definición 18. Los algoritmos de preflujo mantienen un preflujo en cada etapa intermedia. Dado un preflujo, se define el *exceso* de cada nodo $i \in \mathscr{N}$ del siguiente modo:

$$e(i) = \sum_{j \in A(i)} f_{ji} - \sum_{j \in D(i)} f_{ij}. \tag{2.4}$$

Por (2.3), en un preflujo $e(i) \geq 0$ para cada $i \in \mathscr{N} \setminus \{1, n\}$, y como ningún arco sale del nodo n se tiene que $e(n) \geq 0$. Por lo tanto, el nodo origen es el único nodo que puede tener exceso negativo.

Definición 19. Se denomina *nodo activo* a un nodo con exceso estrictamente positivo.

Nótese que los nodos origen y destino nunca están activos.

Este tipo de algoritmos busca lograr la factibilidad, ya que la presencia de nodos activos indica que la solución no es factible. En consecuencia, la operación básica de estos algoritmos es seleccionar un nodo activo e intentar eliminar su exceso empujando el flujo hacia sus nodos vecinos.

Dado un nodo activo, se pretende enviar flujo al nodo destino, de manera que este es empujado a los nodos que están más cerca del n. Esto equivale a empujar el flujo mediante arcos admisibles (estos verifican $d(i) = d(j) + 1$). En caso de que no exista un arco admisible, se ha de aumentar la distancia de etiquetado para poder crear al menos un arco admisible. Por tanto, el algoritmo finalizará cuando la red no contenga ningún nodo activo.

Notación. Cada vez que el algoritmo realiza un empuje, se denota como δ a las unidades de flujo que se realizan en dicho empuje desde un nodo i a un nodo j.

Definición 20. Dado un empuje de δ unidades de flujo de i a j se disminuye tanto el exceso del nodo i, $e(i)$, como r_{ij} en δ unidades y también aumentan $e(j)$ y r_{ji} en δ unidades. Así pues, dado un empuje de δ unidades de flujo en un arco (i, j), se dice que *es saturante* si $\delta = r_{ij}$ y en caso contrario se dice que el arco es *no saturante*.

Cabe destacar que los empujes no saturantes en el nodo i reducirán $e(i)$ a cero.

Los anteriores resultados pueden consultarse con más detalle en [1] y en [3, pág. 736-741].

El procedimiento de los algoritmos de preflujo es el siguiente. Inicialmente se realiza una *operación de preproceso* que consiste en saturar los arcos que salen del nodo 1 y así generar nodos activos en sus extremos. Así pues, se asignan los excesos de flujo a los nodos. Se asignan distancias nulas para todos los nodos salvo para el 1, ya que $d(1) = n$ siendo esta una etiqueta válida. Después, el algoritmo selecciona uno de estos nodos activos. En caso de que ninguno de sus arcos sea admisible, para llevar a cabo el empuje, es necesario aumentar su etiqueta de distancia, a esta operación se le conoce como *operación de reetiquetado*. Así, una vez que se tienen arcos admisibles se realiza el empuje de flujo por dichos arcos. El valor de este empuje se corresponde con el mínimo valor entre el exceso del nodo activo seleccionado y la capacidad residual correspondiente al arco admisible de dicho nodo.

Posteriormente, se selecciona un nodo activo, si hay arcos admisibles se realiza un nuevo empuje, pero en caso de que no haya arcos admisibles el algoritmo realiza una nueva operación de etiquetado.

Del mismo modo se procede hasta que el exceso de flujo resida en el nodo origen o destino, lo cual implicaría que el preflujo actual sería un flujo. Por tanto, se finaliza el algoritmo y, además, se tendrá que el exceso que reside en el destino es el valor del flujo máximo.

Véase formalmente con el siguiente resultado.

Teorema 2.3. *Suponer que el algoritmo de preflujo termina y todas las etiquetas de distancia son finitas. Entonces dicho preflujo es el flujo máximo.*

Demostración. Si el algoritmo finaliza y todas las etiquetas de distancia son finitas, se tiene que todos los nodos en $N \setminus \{1,n\}$ tienen exceso cero, ya que no hay nodos activos. Así pues, f es un flujo. Debido a que un flujo f es un flujo máximo si no hay trayectorias de aumento, es decir, n no es accesible desde 1 en $\mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$, por los Lemas 2.1 y 1.3 ese flujo es máximo. □

Este resultado teórico puede consultarse con más detalle en [7, pág. 18-23].

A continuación se muestra un esquema básico del pseudocódigo del algoritmo.

Algorithm 2: Algoritmo de preflujo genérico.

Data: $G, 1, n, f, u$
foreach *arco* $(i,j) \in \mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$ **do**
  $f = 0$;
  $d(i) = 0 \; \forall \, i \neq 1$, $d(1) = n$;
  $f_{1j} = u_{1j}$ para cada arco $(1,j) \in D(1)$, $e(j) = u_{1j} = l_{1j}$;
end
while *la red contiene un nodo activo* **do**
  Seleccionar el nodo activo i;
  if $\exists \, (i,j) \in \mathscr{A}_R$ *admisible* **then**
    empujar $\delta = min\{e(i), r_{ij}\}$ unidades de flujo del nodo i al j;
    actualizar $e(i), e(j), r_{ij}$ y r_{ji};
  end
  if i *activo y* $\nexists$ *arcos admisibles* **then**
    reemplazar $d(i)$ por $min\{d(j)+1 \mid (i,j) \in \mathscr{A}_R, r_{ij} > 0\}$;
  end
end

Véase el mismo ejemplo del capítulo anterior para observar el funcionamiento de los algoritmos de preflujo.

Ejemplo 2. Se parte de la siguiente red.

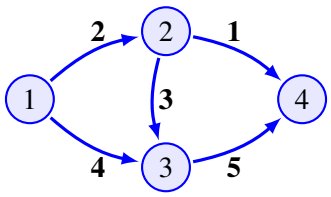

Figura 2.1: Grafo inicial del problema.

En las siguientes iteraciones junto cada nodo podrán observarse las etiquetas de distancia y los excesos correspondientes a cada uno de ellos.

- En primer lugar se realiza la operación de preproceso, consiste en saturar los arcos que salen de 1 y generar nodos activos en sus extremos, así se asignan excesos de flujo positivo a los nodos. Además, este asigna distancias nulas a todos los nodos salvo para 1, ya que $d(1) = 4$. La Figura 2.2 izquierda se corresponde con dicha operación.

- Ahora, se selecciona el nodo 2. Como ninguno de sus arcos son admisibles, entonces se actualizan las distancias. Seleccionando el nodo 3 ocurre lo mismo y se actualizan también las etiquetas de distancia. Este paso se corresponde con la Figura 2.2 derecha.

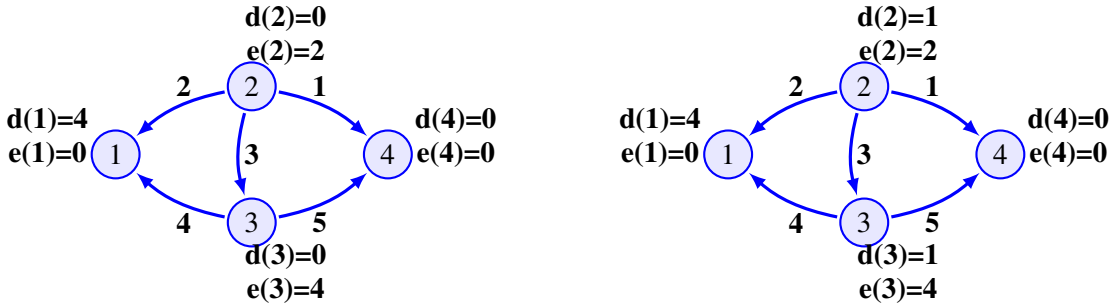

Figura 2.2: A la izquierda puede observarse la red tras la operación de preproceso. A la derecha se tiene la red tras la primera etiqueta.

- A continuación, se selecciona el nodo 2 que es activo y además el arco $(2,4)$ es admisible, ya que $d(2) = d(4)+1$. El arco $(2,3)$ no es admisible, ya que no verifica que $d(2) = d(3)+1$, luego solo realizará un empuje por el arco $(2,4)$. El algoritmo realiza un empuje de $\delta = min\{e(2), r_{24}\} = min\{2,1\} = 1$ unidad de flujo. Por tanto, el arco $(2,4)$ queda saturado. La Figura 2.3 izquierda se corresponde con este paso.

- Se realiza una nueva operación de etiquetado, ya que no hay arcos admisibles. En este caso se actualiza la etiqueta del nodo 2, $d(2) = min\{d(3)+1, d(1)+1\} = min\{2,5\} = 2$. Esta actualización de etiquetado se corresponde con la Figura 2.3 derecha.

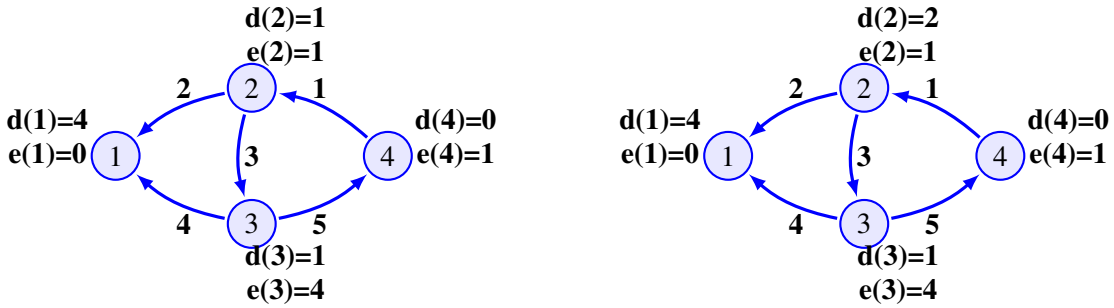

Figura 2.3: A la izquierda se observa la red tras el empuje saturante. A la derecha, la red con la nueva operación de etiquetado.

- Ahora, en la red hay un nodo activo, el 3, y un arco admisible para poder realizar un empuje. Se selecciona el nodo 3, ya que el arco $(3,4)$ es admisible porque verifica $d(3) = d(4)+1$. Se realiza un empuje de $\delta = min\{e(3), r_{34}\} = min\{4,5\} = 4$ unidades de flujo. En este caso el arco que une 3 y 4 no queda saturado. La Figura 2.4 izquierda se corresponde con este empuje.

- El nodo 3 ya no es activo. Se busca otro nodo activo en la red, el 2. El arco $(2,3)$ es admisible, ya que $d(2) = d(3)+1$, luego se realiza un empuje de $\delta = min\{e(2), r_{23}\} = min\{1,3\} = 1$ unidad de flujo. Así, el arco $(2,3)$ tampoco se satura. La Figura 2.4 derecha se corresponde con este paso.

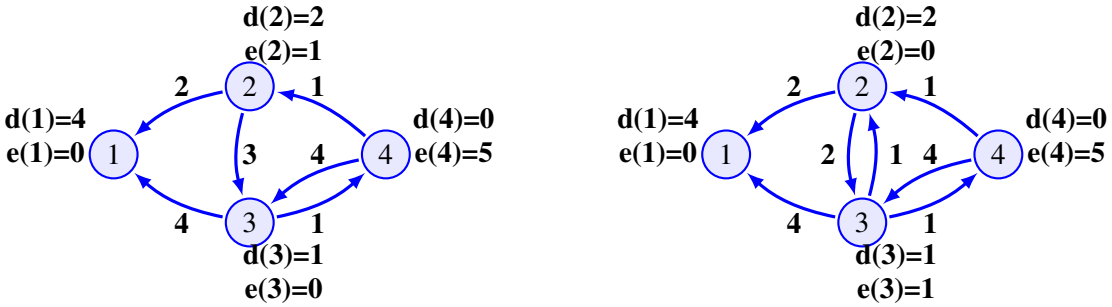

Figura 2.4: A la izquierda se observa la red tras el empuje realizado por el nodo 3. A la derecha, la red tras el empuje realizado por 2.

- Ahora, el nodo 3 es activo, ya que tiene un exceso de flujo positivo. Además, el arco $(3,4)$ es admisible ya que $d(3) = d(4)+1$. Se realiza un empuje de $\delta = \{e(3), r_{34}\} = \{1,1\} = 1$ unidad de flujo, por lo que el arco $(3,4)$ queda saturado.

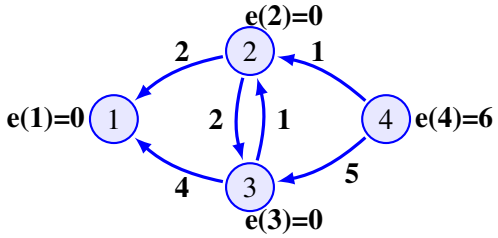

Figura 2.5: Red residual resultante tras el último empuje realizado por el nodo 3.

En la red no hay más nodos activos ni arcos admisibles. Como $d(1) = 4$, la red residual no contiene ninguna cadena del nodo 1 a 4, por el Lema 2.1 el algoritmo finaliza.

Además, el exceso de flujo reside en el nodo 4, el nodo destino, por lo que este preflujo es un flujo. Por el Teorema 2.3 el valor de este flujo, $e(4) = 6$, es el flujo máximo.

2.3. Corrección del algoritmo

Cabe destacar que las etiquetas de distancia son siempre válidas y el número de veces que se incrementan es finito. El hecho de que las etiquetas sean siempre válidas es gracias a que estos algoritmos empujan el flujo solo en arcos admisibles y se vuelve a etiquetar un nodo solo cuando no posee ningún arco admisible. Por otro lado, que el número de veces que se incrementan las etiquetas sea finito se debe al siguiente resultado.

Lema 2.4. *En cualquier etapa del algoritmo cada nodo i con exceso positivo está conectado al nodo origen por una trayectoria dirigida (cadena) desde el nodo i al nodo origen en la red residual $\mathscr{G}_R = (\mathscr{N}, \mathscr{A}_R)$.*

Demostración. Sea f un preflujo, se tiene que $e(1) \leq 0$ y $e(i) \geq 0 \ \forall \ i \in \mathscr{N} \setminus \{1\}$. Por el Teorema de descomposición del flujo[1] se puede descomponer cualquier preflujo con respecto a la red original $\mathscr{G} =$

[1]Cada flujo de una trayectoria y ciclo tiene una única representación como flujo de arco no negativo. A la inversa, cada flujo no negativo f se puede representar como un flujo de trayectoria y ciclo (no necesariamente único) con las propiedades:

1. Cada trayectoria dirigida con flujo positivo conecta un nodo con déficit con un nodo con exceso.
2. A lo sumo $n+m$ trayectorias y ciclos tienen un flujo distinto de cero; fuera de estos, a lo sumo m ciclos tienen flujo distinto de cero.

$(\mathscr{N},\mathscr{A})$ en flujos no negativos a lo largo de trayectorias desde el origen hasta el destino, trayectorias desde el nodo origen hasta los nodos activos, y flujos alrededor de ciclos dirigidos. Sea i un nodo activo en relación al preflujo en $\mathscr{G}=(\mathscr{N},\mathscr{A})$. La descomposición del flujo de f ha de contener una trayectoria P desde el origen hasta el nodo i, ya que las trayectorias del nodo origen al destino y los flujos alrededor de los ciclos no contribuyen al exceso del nodo i. Así, la red residual contiene la inversión de P (es la trayectoria P con la orientación de sus arcos invertida), por lo que hay una trayectoria dirigida desde el nodo i al origen. □

Este resultado implica que durante la operación de re-actualización de etiquetas el algoritmo no se minimiza sobre un conjunto vacío.

Lema 2.5. *Para cada nodo $i\in\mathscr{N}$ se tiene que $d(i)<2n$.*

Demostración. Debido a que la última vez que el algoritmo actualizase la etiqueta del nodo i, este tendría un exceso positivo, por lo que la red residual contendría una cadena P de longitud a lo sumo $n-2$ desde el nodo i al nodo origen. El hecho de que $d(1)=n$ y que $d(i)\leq d(j)+1$ para cada arco (i,j) en la cadena P implica que $d(i)\leq d(1)+\mid P\mid<2n$. □

Cada vez que al algoritmo vuelve a etiquetar el nodo i, $d(i)$ aumenta en al menos 1 unidad, así pues se obtiene el siguiente resultado.

Lema 2.6. *Cada etiqueta de distancia aumenta a lo sumo $2n$ veces. En consecuencia, el número total de operaciones de reetiquetado es a lo sumo $2n^2$.*

Lema 2.7. *Si el algoritmo reetiqueta cualquier nodo a lo sumo k veces, el algoritmo satura los arcos (reduce la capacidad residual a cero) al menos $\frac{km}{2}$ veces.*

Demostración. Entre dos saturaciones consecutivas de un arco (i,j), tanto $d(i)$ como $d(j)$ aumentan al menos dos unidades. Según la hipótesis, el algoritmo aumenta cada etiqueta de distancia a lo sumo k veces, esto implicaría que el algoritmo podría saturar cualquier arco como máximo $\frac{k}{2}$ veces. Luego, el número total de saturaciones de los arcos sería $\frac{km}{2}$, lo cual afirma el lema. □

Como resultado del anterior Lema y del Lema 2.5 se tiene la siguiente afirmación.

Lema 2.8. *El algoritmo realiza a lo sumo nm empujes saturantes.*

Demostración. Supongase que un aumento satura un arco (i,j), en este primer envío saturante $d(i)+d(j)\geq 1$. En el último empuje saturante que produzca $d^*(i)+d^*(j)=d^*(j)+1+d^*(j)=2d^*(j)+1\leq 4n$, por el Lema 2.5.

Como entre dos envíos saturantes $d(i)+d(j)$ aumentan en al menos 1 cada distancia, empieza en 1 y acaba en $4n$. A lo sumo se hacen n envíos y como hay m arcos, se tiene que se realizan nm empujes saturantes. □

Propiedad 2. Si el algoritmo reetiqueta cualquier nodo k veces, el tiempo total empleado en buscar arcos admisibles y reetiquetar nodos es,

$$O(k\sum_{i\in\mathscr{N}}|\mathscr{A}(i)|)=O(km).$$

En consecuencia del Lema 2.6 y la Propiedad 2 el tiempo total necesario para identificar arcos admisibles y para realizar operaciones de reetiquetado se tiene que es $O(nm)$. A continuación, se cuenta el número de empujes no saturantes que realiza el algoritmo.

Lema 2.9. *El algoritmo de preflujo genérico realiza $O(n^2m)$ empujes no saturantes.*

Demostración. Sea I el conjunto de nodos activos y la función potencial[2]:

$$\Phi = \sum_{i \in I} d(i).$$

Como $\mid I \mid < n$ y $d(i) < 2n \; \forall \; i \in I$, el valor inicial de Φ (tras la operación de preproceso) a lo sumo es $2n^2$. Una vez que el algoritmo finaliza Φ es cero.

Durante la operación de empuje o reetiquetado de un nodo i ha de aplicarse uno de los siguientes casos.

1. El algoritmo es incapaz de encontrar un arco admisible en el cual impulsar el flujo. En este caso, la etiqueta de distancia del nodo i aumentará en $\epsilon \geq 1$ unidades. Así pues, esta operación incrementa Φ a lo sumo ϵ unidades. Como el aumento total en $d(i)$ para cada nodo i a lo largo de la ejecución del algoritmo está limitado por $2n$, el total de incrementos en Φ debido a los aumentos de las etiquetas de distancia está limitado por $2n^2$.

2. El algoritmo identifica un arco por el que se puede empujar flujo, por lo que realiza un empuje saturante o no saturante. Un empuje saturante en el arco (i, j) podría crear un nuevo exceso en el nodo j, aumentando así el número de nodos activos en 1, y aumentando Φ en $d(j)$, que podría ser como mucho $2n$ por cada empuje saturante. Luego, podría aumentar $2n^2m$ sobre todos los empujes saturantes.

   Sin embargo, un empuje no saturante puede disminuir Φ en $d(i)$ ya que i se convertirá en un nodo inactivo, pero simultáneamente Φ aumentará en $d(j) = d(i) - 1$ si el empuje hace que el nodo j se convierta en activo, por lo que la disminución total de Φ será de valor 1. En caso de que el nodo j estuviese activo antes del empuje, Φ disminuye en una cantidad de $d(i)$. Consecuentemente, la disminución neta de Φ es de al menos una unidad por empuje no saturante.

En conclusión, el valor inicial de Φ es a lo sumo $2n^2$ y el máximo incremento posible en Φ es $2n^2 + 2n^2m$. Cada empuje no saturante disminuye Φ en al menos una unidad y Φ siempre permanece no negativo. Luego, el algoritmo solo puede realizar a lo sumo $2n^2 + 2n^2 + 2n^2m = O(n^2m)$ empujes no saturantes. □

Hay varias estructuras de datos disponibles para almacenar las listas de modo que el algoritmo pueda agregar, eliminar o seleccionar elementos de ella en un tiempo $O(1)$. Como consecuencia, es fácil implementar el algoritmo en un tiempo $O(n^2m)$, por lo que se tiene el siguiente teorema.

Teorema 2.10. *El algoritmo de preflujo genérico se ejecuta en tiempo $O(n^2m)$.*

Los anteriores resultados teóricos pueden consultarse con más detalle en [1].

2.4. Modificaciones para los algoritmos de preflujo

Existen modificaciones para los algoritmos de preflujo que pueden mejorar su rendimiento.

Como se ha visto, estos algoritmos realizan operaciones de empuje y reetiquetado en nodos activos hasta que todo el exceso de flujo llega al nodo destino o vuelve al nodo origen. El algoritmo establece un preflujo máximo[3] antes de que se establezca un flujo máximo. Por lo que las operaciones de empuje y etiquetado aumentan las etiquetas de distancia de los nodos activos hasta que son superiores a n para que los excesos puedan regresar al origen.

Una posible modificación de este algoritmo es mantener un conjunto de nodos $\mathcal{N}'$ que satisfaga la propiedad de que la red residual no contiene ninguna cadena desde un nodo en $\mathcal{N}'$ hasta el nodo

[2]Se usan las técnicas de la función potencial para establecer la complejidad de un algoritmo mediante el análisis de los efectos de distintas operaciones en una función adecuadamente definida. El uso de la función potencial permite definir una relación "contable" entre las ocurrencias de varias operaciones de un algoritmo para poder obtener un límite en las operaciones que pueden ser difíciles de obtener usando otros argumentos.

[3]Se hace referencia a un preflujo máximo como preflujo con el máximo flujo posible en el destino.

destino. Al comenzar este proceso, $\mathscr{N}' = 1$, pero posteriormente, cuando la etiqueta de distancia de un nodo sea mayor o igual que n, se añadirá a $\mathscr{N}'$. Además para los nodos de $\mathscr{N}'$ no se realizan operaciones de empuje y reetiquetado, por lo que el algoritmo finaliza cuando todos los nodos de $\mathscr{N} - \mathscr{N}'$ están inactivos. En conclusión, el preflujo actual es óptimo, luego, el preflujo máximo se convierte en un flujo máximo.

Para agregar un nodo j a $\mathscr{N}'$ una condición suficiente es que $d(j) \geq n$, aunque este enfoque no es muy efectivo ya que no reduce significativamente el tiempo de ejecución del algoritmo. Otro método sería realizar una búsqueda inversa de amplitud de la red residual para obtener etiquetas de distancia exactas y agregar todos los nodos que no tengan ninguna cadena al destino a $\mathscr{N}'$. Realizar esta búsqueda después de αn operaciones de reetiquetado para alguna constante α no afecta a la complejidad de estos algoritmos pero mejora el comportamiento empírico del algoritmo.

Cabe destacar la flexibilidad y el potencial que tienen estos algoritmos para realizar nuevas mejoras de ellos mismos. Además, cuando se especifican diferentes reglas para seleccionar nodos activos para efectuar las operaciones de empuje y reetiquetado pueden derivarse muchos algoritmos diferentes, cada uno con una complejidad diferente.

Los algoritmos de preflujo genéricos poseen una operación denominada "cuello de botella" que consiste en realizar envíos o empujes no saturantes. Para ello, existen diferentes formas de examinar los nodos activos que producen disminuciones en el número de empujes no saturantes, con lo cual se mejora el algoritmo. Algunos de los métodos que se aplican a estos algoritmos son los siguientes.

1. *Algoritmo de preflujo FIFO*. Estos examinan los nodos activos en orden desde el primero que entra y primero que sale. Este algoritmo consigue ejecutarse en un tiempo $O(n^3)$.

2. *Algoritmo de preflujo de distancia máxima*. Este algoritmo siempre realiza empujes desde un nodo activo con el mayor valor de etiqueta de distancia. El algoritmo se ejecuta en un tiempo $O(n^2 m^{1/2})$, el cual es mejor que el visto anteriormente.

3. *Algoritmo de exceso de escala*. Este algoritmo empuja el flujo desde un nodo con demasiado exceso a un nodo con muy poco exceso. Estos algoritmos se ejecutan en un tiempo $O(nm + n^2 logU)$.

Los límites de tiempo para estos algoritmos de empuje son ajustados, salvo para el tercer tipo. Es decir, para algunas clases de redes, los dos primeros tipos de algoritmos, en el peor de los casos realizan tantos cálculos como indican sus límites de tiempo. Lo cual indica que no se pueden mejorar los límites de tiempo de estos mediante un análisis más inteligente.

A continuación, se va a mostrar con más detalle uno de estos algoritmos. Se puede consultar información acerca del resto en [1].

2.4.1. Algoritmos de preflujo FIFO

El desarrollo de los algoritmos de preflujo FIFO es el siguiente.

Si se selecciona el nodo i y se realiza un empuje saturante, este todavía podría estar activo, pero no es necesario que el algoritmo de preflujo vuelva a seleccionar este nodo en la siguiente iteración. Así pues, puede seleccionarse otro nodo para la siguiente operación de envío de flujo o reetiquetado.

Es fácil considerar como regla que cada vez que el algoritmo seleccione un nodo activo, este continúe enviando el flujo desde ese nodo hasta que su exceso se vuelva cero o el algoritmo reetiquete el nodo. Como consecuencia, podrían realizarse varios empujes saturantes seguidos por un empuje no saturante o una operación de reetiquetado. Esta secuencia de operaciones se conoce como *examen de nodo*. Por tanto, para la selección de nodos en este tipo de algoritmos se considera esta regla.

Los algoritmos de preflujo FIFO examinan los nodos activos en el orden FIFO, es decir, el primer nodo que entra será el primer nodo que sale. Se mantiene la "lista" como una cola y se selecciona el nodo i del principio de la lista, se realiza un empuje desde dicho nodo y se agregan los nuevos nodos activos a la lista. Este algoritmo sigue analizando el nodo i hasta que este quede inactivo o se reetiquete.

En caso de reetiquetarlo, se agrega el nodo i a la parte final de la cola. Por tanto, el algoritmo finalizará cuando la cola de nodos activos este vacía.

En cuanto a la complejidad computacional de este tipo de algoritmos, el caso más desfavorable puede encontrarse cuando se divide el número total de exámenes de nodos en diferentes fases. La primera fase consiste en examinar los nodos que se activan durante la operación de preproceso. En la segunda fase se examinan los nodos que están en la cola después de que el algoritmo haya examinado los nodos en la primera fase. Igualmente, en la tercera fase se observan los nodos que están en la cola después de que el algoritmo haya examinado los nodos en la segunda fase, y así sucesivamente.

En general, el algoritmo realiza $2n^2+n$ fases. Cada una de estas fases examina como máximo un nodo una vez y cada examen de nodo realiza a lo sumo un empuje no saturante. Así, el límite $2n^2+n$ en el número total de fases implicaría un límite de $O(n^3)$ en el número de empujes no saturantes. Por tanto, este resultado implica que los algoritmos de preflujo FIFO se ejecuten en un tiempo $O(n^3)$ ya que la operación de cuello de botella de estos es el número de empujes no saturantes.

En cuanto al límite de fases del algoritmo, se considera el cambio total en la función potencial $\Phi = max\{d(i) \mid i \text{ es nodo activo}\}$ durante toda una fase. El "cambio total" se entiende como la diferencia entre los valores iniciales y finales de la función potencial durante una fase. Se consideran los dos casos siguientes.

1. El algoritmo realiza al menos una operación de reetiquetado durante una fase. Entonces Φ podría aumentar tanto como el máximo aumento en cualquier etiqueta de distancia. Además, el Lema 2.6 implica que el aumento total de todas las fases de Φ es a lo sumo $2n^2$.

2. El algoritmo no realiza operaciones de reetiquetado durante una fase. En este caso el exceso de cada nodo que estaba activo al comienzo de la fase mueve los nodos que tienen pequeño valor como etiqueta de distancia. Como consecuencia, se tiene que Φ disminuye al menos una unidad.

Por tanto, combinando los casos anteriores, el número total de fases es a lo sumo $2n^2+n$, donde el segundo término se corresponde con el valor inicial de Φ, el cuál como máximo puede ser n. Así pues se puede enunciar el siguiente teorema.

Teorema 2.11. *Los algoritmos de preflujo con la regla FIFO se realizan en un tiempo $O(n^3)$.*

Puede encontrarse más información acerca de los algoritmos de preflujo FIFO en [1].

Capítulo 3

Estudio experimental y conclusiones

En este capítulo se realiza un estudio para comparar el comportamiento de los dos algoritmos propuestos.

El experimento se ha llevado a cabo en un ordenador Acer con procesador Intel Core i5 bajo el lenguaje de programación Java [13] y bajo el entorno de trabajo NetBeans [12].

Los códigos utilizados se han obtenido de [9] y [10], y se han adaptado y ajustado para que respondan a las versiones propuestas en esta memoria y posteriormente se han traducido a Java, desarrollando las funciones necesarias de lectura de ficheros de datos y de escritura de resultados.

Para llevar a cabo el estudio, se han generado problemas de máximo flujo con diferentes configuraciones, estos originalmente están compuestos por 10 nodos origen y 10 nodos destino. Se presentan variaciones en el número de arcos, ya que se han creado grafos con 1000, 2000, 5000 y 10000 arcos añadidos. Además, dentro de estos se distinguen tres casos, ya que se generan grafos en los que la máxima capacidad de flujo que pueden contener sus arcos está representada por variables $U(0,25)$, $U(0,50)$ y $U(0,100)$. Y, también se estudian según la variación de oferta total, se distinguen casos en los que se ofertan 5000, 10000 y 15000 unidades de flujo. Para cada configuración se han generado 30 problemas distintos.

Los ficheros con estos problemas se han creado con una modificación del generador GRIDGEN [2], proporcionada por el director de este trabajo.

Los algoritmos se han aplicado a cada problema bajo las mismas condiciones, se ha registrado su tiempo de ejecución en milisegundos y el flujo máximo obtenido (para comprobar el correcto comportamiento).

Posteriormente, los datos obtenidos se han incorporado a un fichero de datos de R [14] para poder realizar cómodamente los cálculos necesarios.

A continuación se mostrarán tres cuadros, según la capacidad máxima de arco, donde se recogen las medias de tiempo (en ms) de la ejecución de ambos algoritmos diferenciadas según su oferta total y la cantidad de arcos disponibles. Para simplificar notación se hace referencia a FF como el algoritmo de Ford y Fulkerson, y a PF como el algoritmo de preflujo.

Oferta Total	**5000**		**10000**		**15000**	
Método	**FF**	**PF**	**FF**	**PF**	**FF**	**PF**
1000 Arcos	9,17	3,79	14,62	4,31	12,28	4,83
2000 Arcos	14,59	5,38	16,17	5,90	13,45	5,83
5000 Arcos	39,76	4,79	41,86	6,41	43,66	4,28
10000 Arcos	45,24	1,03	94,86	7,52	72,97	4,31

Cuadro 3.1: Cuadro de comparación de tiempos medios con capacidad máxima de los arcos igual a 25.

Oferta Total	5000		10000		15000	
Método	**FF**	**PF**	**FF**	**PF**	**FF**	**PF**
1000 Arcos	8,48	3,59	8,41	3,17	8,83	3,59
2000 Arcos	15,72	4,97	16,97	5,24	16,34	6,34
5000 Arcos	22,28	0,14	46,83	4,55	46,07	4,28
10000 Arcos	21,34	0,41	60,55	1,66	71,55	4,69

Cuadro 3.2: Cuadro de comparación de tiempos medios con capacidad máxima de los arcos igual a 50.

Oferta Total	5000		10000		15000	
Método	**FF**	**PF**	**FF**	**PF**	**FF**	**PF**
1000 Arcos	9,66	4,83	10,76	3,28	11,31	2,66
2000 Arcos	11,24	1,07	17,31	4,90	17,34	4,83
5000 Arcos	11,83	0	24,76	0,55	41	4,93
10000 Arcos	10,76	0	25,86	2,17	29,5	0,55

Cuadro 3.3: Cuadro de comparación de tiempos medios con capacidad máxima de los arcos igual a 100.

A continuación se van a mostrar las gráficas de los tiempos medios de ejecución de cada algoritmo diferenciado según su número de arcos. Se destaca en color azul al algoritmo de Ford y Fulkerson, y en rojo al algoritmo de preflujo.

En estas figuras aparecerán situadas a la izquierda las gráficas correspondientes cuando la capacidad máxima de arco sea una $U(0,25)$, en el centro cuando sea $U(0,50)$, y a la derecha cuando la capacidad máxima de arco sea $U(0,100)$.

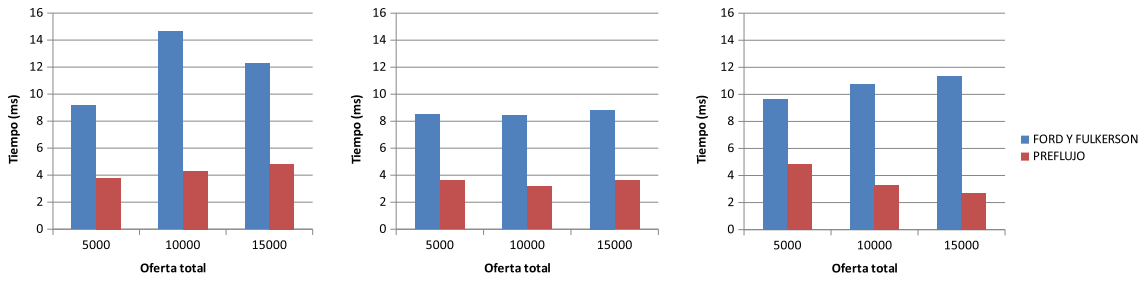

Figura 3.1: Gráficas de tiempo según la oferta total cuando se dispone de 1000 arcos.

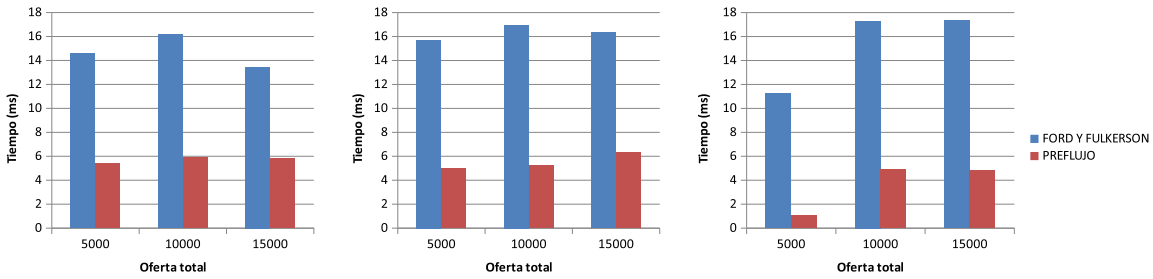

Figura 3.2: Gráficas de tiempo según la oferta total cuando se dispone de 2000 arcos.

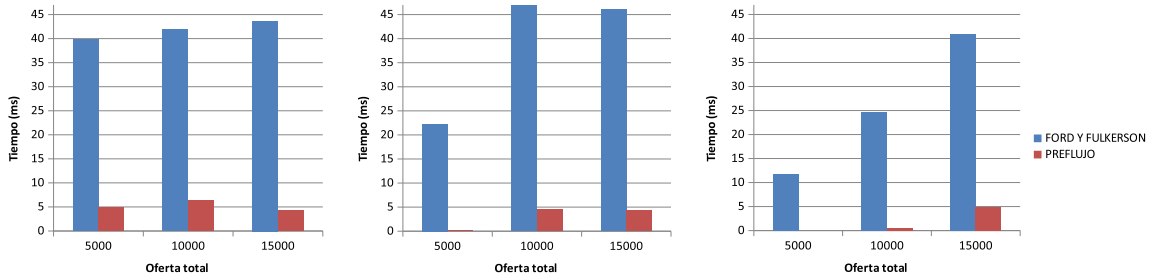

Figura 3.3: Gráficas de tiempo según la oferta total cuando se dispone de 5000 arcos.

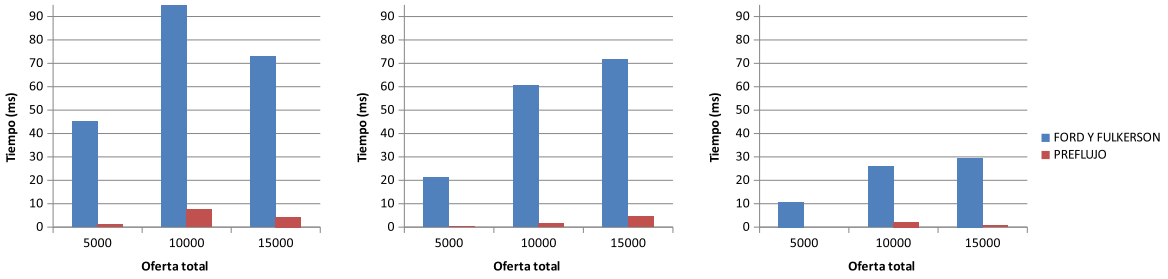

Figura 3.4: Gráficas de tiempo según la oferta total cuando se dispone de 10000 arcos.

En vistas generales de los resultados obtenidos en los anteriores cuadros puede observarse como el algoritmo de preflujo, en el Cuadro 3.1, va de unas 3 hasta unas 44 veces más rápido que el de Ford y Fulkerson (debido a que las mayores variaciones de ambos algoritmos se corresponden cuando se ofertan 5000 unidades de flujo, ya que se recogen datos de 9,17ms y 3,19ms en el caso de 1000 arcos, y 45,24ms y 1,03ms en el caso de 10000 arcos para Ford y Fulkerson y preflujo, respectivamente). Sin embargo, en este caso con los arcos como capacidad neta $U(0,25)$ conforme aumenta la oferta total el algoritmo de preflujo sigue siendo más rápido que el de Ford y Fulkerson, pero rebaja la rapidez con la que lo multiplicaba en el caso citado anteriormente.

En el Cuadro 3.2 cuando la capacidad máxima de arco se corresponde con una $U(0,50)$ en el algoritmo de preflujo la mayor variación de tiempo recorre entre unos 2,5 y unas 50 veces más rápido, aproximadamente, que el algoritmo de Ford y Fulkerson. Además, como en el caso anterior, conforme aumenta la oferta total el algoritmo de preflujo rebaja su tiempo de multiplicación aunque sigue siendo bastante más rápido que Ford y Fulkerson.

En cuanto al Cuadro 3.3 cuya capacidad máxima de arco es una $U(0,100)$ se tienen mejores tiempos de ejecución para ambos algoritmos. Puede observarse como los tiempos de Ford y Fulkerson se han rebajado considerablemente, comparado con los tiempos de los anteriores cuadros citados, ya que el algoritmo tarda menos en buscar trayectorias de aumento debido a que este hará envíos de flujo con mayor capacidad. Sin embargo, para el algoritmo de preflujo el tiempo de mejora de este parece infinito, debido a que es capaz de estudiar algunos de los problemas en un tiempo inferior a la resolución mínima de registro de la función de tiempo (en milisegundos).

En aspectos generales se pueden observar los siguientes hechos para ambos algoritmos.

Algoritmo de Ford y Fulkerson:

- Por un lado al tiempo de ejecución del algoritmo de Ford y Fulkerson le afecta el número de arcos que contiene cada problema, ya que cuantos más arcos más tiempo tarda en obtener la solución. En condiciones generales este algoritmo no es muy estable en cuanto al número de arcos, ya que se observan las variaciones conforme mayor capacidad y número de arcos poseen estos.

- Por otro lado a este algoritmo también le afecta la cantidad de oferta total a distribuir, aunque en menor medida que el número de arcos. Se observa como aumenta el tiempo de ejecución conforme mayores cantidades se disponen de oferta total.

  Sin embargo, cuando se dispone de un número de arcos moderado, en los casos de 1000 y 2000, el incremento de tiempo conforme la oferta total no es muy elevado. Pero, cuando aumenta el número de arcos, en los casos de 5000 y 10000, el tiempo aumenta considerablemente.

- En cuanto a las capacidades máximas de arco, se observa que conforme mayor es la capacidad de los arcos el tiempo tiende a ser menor. Esto es lógico ya que una vez encontrada una trayectoria, esta puede enviar más flujo y finalizar antes.

Algoritmo de preflujo:

- Este algoritmo según el número de arcos, se muestra muy estable. En el caso del Cuadro 3.1, cuando se dispone de una capacidad máxima de arco $U(0,25)$ se tienen tiempos en torno a 5ms, en el caso del Cuadro 3.2 con capacidad $U(0,50)$, los valores se mueven con rangos muy similares al anterior, si bien es cierto que hay un poco más de variación. Aparecen tiempos de 0,14ms y tiempos de 6,36ms. Además, en el Cuadro 3.3 los tiempos todavía se reducen más. Esto es lógico en cierta medida porque el algoritmo no necesita etiquetar cada vez todos los nodos hasta llegar al destino, lo cual no le perjudica en el rendimiento.

- La oferta total de la que se dispone tampoco afecta mucho a estos algoritmos. En condiciones generales se tienen tiempos con rangos similares. Puede observarse, en el caso del Cuadro 3.2, con capacidad máxima de arco $U(0,50)$, como el aumento de oferta total con mayor número de arcos incrementa levemente el tiempo de ejecución.

- En cuanto a las capacidades máximas de arco ocurre lo mismo que antes, a cuanto mayor es la capacidad de los arcos menor es, en general, el tipo de resolución. Esto se debe a que en general conforme se dispone mayor capacidad de arco este algoritmo realiza empujes con mayor cantidad de flujo, así finaliza antes.

En relación a las gráficas presentadas, es fácil observar a simple vista como las barras correspondientes al algoritmo de preflujo siempre están por debajo de las correspondientes al algoritmo de Ford y Fulkerson, pues como se ha visto este siempre es más lento que el de preflujo.

Comparando todas las gráficas mostradas según el tamaño de red, se observa como conforme aumenta el tamaño de la red, es decir conforme mayor número de arcos se disponen, generalmente aumenta también la diferencia de los tiempos de ejecución.

En conclusión, puede garantizarse que para el algoritmo de Ford y Fulkerson es más costoso hallar trayectorias de aumento conforme mayor número de arcos y oferta total tiene, pero menor capacidad en sus arcos. Sin embargo, exceptuando algunos casos puntuales, puede decirse que el algoritmo de preflujo es estable en todos los aspectos.

A continuación, como observación, se muestran los cuadros correspondientes a la desviación típica, según la capacidad máxima de arco, para observar la estabilidad de ambos algoritmos.

Oferta Total	**5000**		**10000**		**15000**	
Método	**FF**	**PF**	**FF**	**PF**	**FF**	**PF**
1000 Arcos	7,85	6,85	13,80	7,11	11,65	7,33
2000 Arcos	4,07	7,55	6,62	7,68	6,88	7,63
5000 Arcos	7,97	7,28	8,47	7,78	7,57	7,06
10000 Arcos	8,65	3,87	36,38	12,96	11,59	7,11

Cuadro 3.4: Desviación típica para los algoritmos con 25 unidades como capacidad máxima de arco.

Oferta Total	**5000**		**10000**		**15000**	
Método	**FF**	**PF**	**FF**	**PF**	**FF**	**PF**
1000 Arcos	7,37	4,32	1,24	4,05	2,7	4,36
2000 Arcos	3,01	3,8	3,49	4,43	3,39	5,7
5000 Arcos	5,13	0,74	11,0	4,1	7,1	4,53
10000 Arcos	5,14	1,24	21,57	7,45	10,02	4,42

Cuadro 3.5: Desviación típica para los algoritmos con 50 unidades como capacidad máxima de arco.

Oferta Total	**5000**		**10000**		**15000**	
Método	**FF**	**PF**	**FF**	**PF**	**FF**	**PF**
1000 Arcos	7,69	7,33	7,36	6,53	7,12	5,92
2000 Arcos	8,2	4,0	6,39	7,43	7,53	7,33
5000 Arcos	6,8	0	7,86	2,97	9,74	7,48
10000 Arcos	7,36	0	11,24	5,53	7,65	2,97

Cuadro 3.6: Desviación típica para los algoritmos con 100 unidades como capacidad máxima de arco.

Una vez examinadas las desviaciones típicas de los tiempos de resolución, Cuadros 3.4, 3.5 y 3.6, se observa como los datos obtenidos son considerablemente elevados, salvo alguna excepción, comparados con la magnitud de los valores medios de tiempo. Esto se debe a la gran diferencia de estructura que puede aparecer en los ficheros de datos, a pesar de que todos están definidos con los mismos parámetros.

En el caso del algoritmo de Ford y Fulkerson estas desviaciones son altas debido a que los datos obtenidos también son grandes.

En el caso del algoritmo de preflujo, salvo algunas excepciones dadas para los casos en los que el algoritmo es más rápido, también se tienen altas desviaciones típicas debido la alta velocidad del algoritmo y la resolución mínima del registro de tiempos, ya que como se ha comentado anteriormente, en este caso aparecen varios problemas con tiempo cero de ejecución.

Tras el estudio realizado, aunque en cuanto a los datos obtenidos con la desviación típica no se muestre gran estabilidad, se puede concluir que el algoritmo de preflujo es más eficaz que el algoritmo de Ford y Fulkerson, ya que no le afectan en la misma medida las variaciones del número de arcos, oferta y capacidad.

Bibliografía

[1] R. K. AHUJA, T.L. MAGNANTI Y J. B. ORLIN, *Network flows: theory, algorithms and applications*, New Jersey, 1993.

[2] D. P. BERTSEKAS, *Linear network optimization. Algorithms and codes*, MIT Press, Massachusetts, 1991.

[3] T.H. CORMEN, C.E. LEISERSON, R.L. RIVEST Y C. STEIN, *Introduction to Algorithms*, MIT Press, Massachusetts, tercera edición, 2009.

[4] L. R. FORD JR Y D. R. FULKERSON, *Maximal flow through a network*, Canadian Journal of Mathematics, **8** (1956), 399–404.

[5] L.R. FORD, JR. Y D.R. FULKERSON, *Flows in Networks*, Princeton University Press, 1962.

[6] A. GARCÍA OLAVERRI, *Teoría de grafos*, Universidad de Zaragoza, 2014.

[7] A. V. GOLDBERG, *Efficient graph algorithms for sequential and parallel computers*, Massachusetts Institute of Technology, Massachusetts, 1987.

[8] A. V. GOLDBERG Y R. E. TARJAN, *A New Approach to the Maximum-Flow Problem*, Journal of the Association for Computing Machinery, **35** (4) (1988), 921–940.

[9] M. KUMAR BHOJASIA, *Sanfoundry Technology Education Blog*, `https://www.sanfoundry.com/java-program-implement-ford-fulkerson-algorithm/`.

[10] A. NAUMENKO, *Algorithms and Data Structures*, `https://sites.google.com/site/indy256/algo/preflow`.

[11] K. WAYNE, *Theory of Algorithms, Maximum Flow Applications*, Princeton University, 2001.

[12] *Apache NetBeans*, `https://netbeans.org/`.

[13] *Java*, `https://www.java.com/es/about/whatis_java.jsp`.

[14] *The R Project for Statistical Computing*, `https://www.r-project.org/`.

Apéndice A

Modelado de los problemas del apartado 1.1 mediante máximo flujo

En esta sección puede consultarse como se modelan los problemas enunciados anteriormente en la sección 1.1, aplicaciones del problema de flujo máximo, ya que ahora se dispone de los elementos teóricos necesarios para llevarlos a cabo.

A.1. Problema de flujo factible

Como se ha comentado anteriormente, este tipo de problemas busca determinar si existe un envío con flujo factible, es decir, que los demandantes reciban las cantidades que solicitan.

Entre los ofertantes y demandantes existe una red de conexiones con distintas capacidades de flujo como paso. Este problema de factibilidad, 1.1.1, se puede resolver como problema de flujo máximo definiendo una red aumentada. Para ello, se introducen dos nuevos nodos en la red, un nodo origen y un nodo destino (se corresponde con el primer caso enunciado en la sección 1.4). Estos nodos se conectan con los correspondientes nodos en la red mediante arcos, asociándoles una determinada capacidad máxima que represente la oferta o demanda, es decir, los nodos que se unen con el origen y destino son los nodos que ofertan o demandan producto. Además, en estos problemas para los nodos que ofertan y demandan se verifica que la suma total de la oferta y la demanda es cero, para que el problema sea equilibrado y así poder lograr la factibilidad.

El modo de conectar dichos nodos con el origen y destino es el siguiente. Se unirán los nodos ofertantes, cuya capacidad es positiva, con el nodo origen y se asocia dicha capacidad a su arco correspondiente. Por otro lado, los nodos demandantes poseen capacidad negativa. Estos se unen con el nodo destino y se asocia a sus arcos correspondientes dicha capacidad en positivo.

Así, una vez conectados con los nodos origen y destino, esta oferta y demanda pasa a ser la capacidad que conecta dichos nodos con el origen y destino.

Considerando esta red aumentada es posible resolver el problema como problema de flujo máximo, y así se obtiene el flujo máximo del problema inicial.

Para este tipo de problemas en caso de que el flujo máximo saturase todos los arcos que finalizan en el destino, se dice que la solución es factible. Por tanto, la red original tiene un flujo factible si y solo si la red transformada tiene un flujo que satura todos los arcos que finalizan en el destino. Es decir, en este caso se tendría un corte $(X, \bar{X})$ de la forma $\bar{X} = \{n\}$ y $X = \mathcal{N} \setminus \{n\}$.

Veáse ahora el ejemplo de los puertos marítimos.

Se dispone de una mercancía disponible y esta es ofertada por algunos puertos y demandada por otros. Los nodos representan los puertos marítimos, estos están asociados a una determinada oferta/demanda de dicha mercancía. Por tanto, los arcos representan los posibles envíos de la mercancía con una determinada capacidad que se representa junto a cada arco. La cuestión es ver si se pueden satisfacer todas las demandas utilizando los suministros disponibles.

Originalmente se tiene la siguiente red donde sobre los nodos puede observarse la oferta y demanda (en color rojo), y sobre los arcos se observa la cantidad de mercancía que puede enviarse entre dichos puertos.

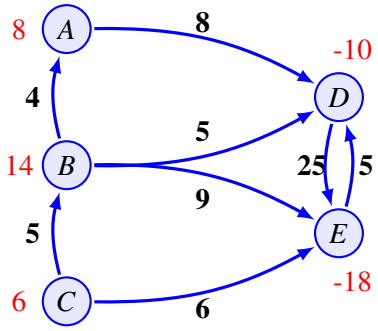

Figura A.1: Red inicial del problema.

Para resolver el problema, ha de aumentarse la red, para ello se crea el nodo origen y destino. Estos se unen con los nodos correspondientes, asociándoles como capacidad la oferta y demanda correspondiente a cada nodo.

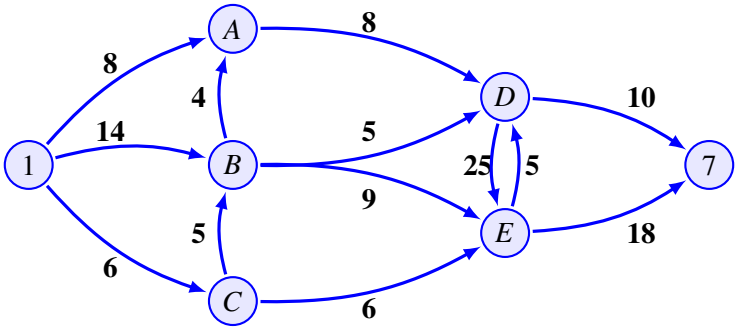

Figura A.2: Red aumentada del problema.

El actual problema ya puede resolverse como problema de flujo máximo.

Una vez resuelto, se tiene que el máximo flujo es 28, este coincide con la cantidad ofertada y demandada. Así, todos los arcos correspondientes al nodo origen y nodo destino se han saturado, lo cual implica que el flujo obtenido es factible. Es decir, en este caso todos los demandantes obtienen las cantidades de producto solicitadas.

A.2. Problema de los representantes

Estos problemas pretenden encontrar una situación de "equilibrio" de modo que a cada representante se le asigne una determinada actividad o trabajo. Para ello se amplia la red, se unen los nodos correspondientes con un nodo origen y un nodo destino. Además, en este problema todos los arcos poseen capacidad unitaria, salvo los que se unen con el destino que poseen una capacidad "u_k". Aunque pueden encontrarse variantes en cuanto a la capacidad de los arcos según la cantidad de representantes que se requieran en el problema.

En el ejemplo propuesto en la sección 1.1.2 se hace referencia a una ciudad en la que hay ciudadanos que residen en ella, estos de denotan como $C_1, ...C_r$, un conjunto de asociaciones vecinales denotadas como $A_1, ..., A_q$, y un conjunto de partidos políticos denotados como $P_1, ..., P_p$. Cada ciudadano es miembro de al menos una asociación vecinal, pero solo puede pertenecer a un partido político. Por tanto, cada asociación vecinal debe de asignar a uno de sus ciudadanos para que sea representado en el consejo de gobierno de la ciudad. Así, el número de miembros que pertenezcan al partido político ha de ser como

máximo u_k. Por tanto, se quiere ver si es posible encontrar un consejo que satisfaga las propiedades de equilibrio.

Cabe destacar que la capacidad de todos los arcos es unitaria, ya que se pretende que un ciudadano no sea representado por más de un partido político y este sea miembro de al menos una asociación vecinal. Supongamos que dadas 3 asociaciones vecinales ($q = 3$), 5 ciudadanos ($r = 5$) y 2 partidos políticos ($p = 2$) se estaría ante la siguiente situación.

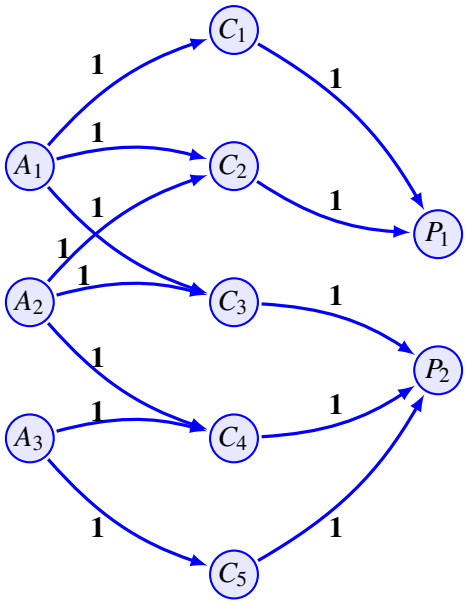

Figura A.3: Red inicial del problema.

Para resolver el problema como problema de flujo máximo, se añade el nodo origen, 1, el cual se conecta con cada tipo de asociación vecinal y se le asigna capacidad de arco 1, ya que se pretende que en el consejo de gobierno de la ciudad haya una representación de cada asociación vecinal. Se añade también el nodo destino, 12, el cuál se une a cada partido político y se le asigna una capacidad de arco correspondiente al máximo número de miembros que pueden pertenecer a cada partido político.

Por tanto, la red transformada quedará del siguiente modo.

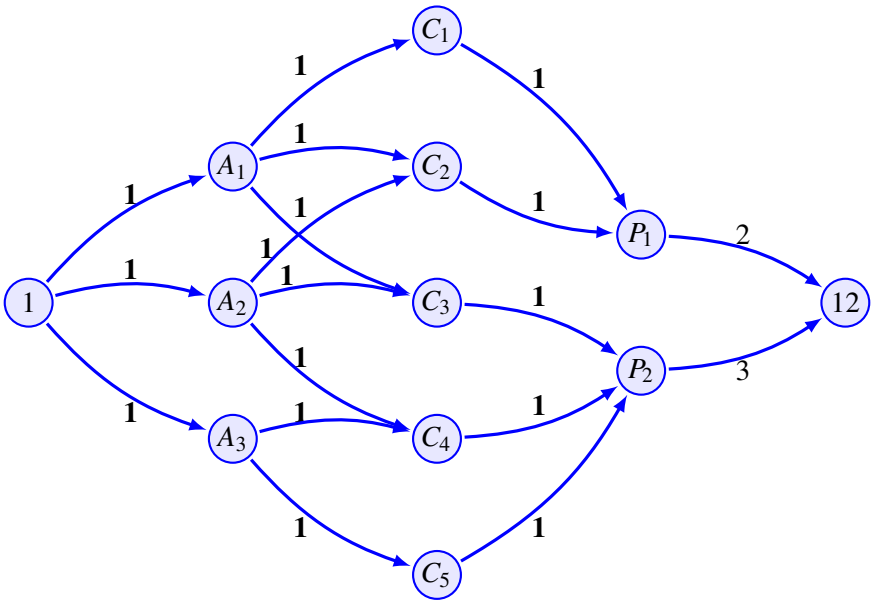

Figura A.4: Red transformada del problema.

Ahora puede resolverse el problema como problema de máximo flujo. Tras buscar todas las posibles trayectorias de aumento, se tiene que el máximo flujo es 3, que coincide con el número de asociaciones vecinales de la ciudad. Por tanto se tendrá un consejo equilibrado, ya que habrá representación de cada una de ellas en el consejo de gobierno de la ciudad.

A.3. Problema de redondeo de matrices

Este tipo de problema se puede utilizar para anonimizar tablas de datos. Por ejemplo, en tablas correspondientes a encuestas tal que, en función de sus datos se podría llegar a identificar algún individuo. Debido a este hecho, el problema consiste en modificar los datos proporcionados originalmente de modo que estos proporcionen valores similares.

Dada una matriz de números reales, para construir la red se asocia a cada arco una capacidad correspondiente a su parte entera y la parte entera +1. El problema de redondeo de matrices requiere que se redondeen los elementos de la matriz, sumas de filas y columnas de modo que la suma de los elementos redondeados en cada fila sea igual a la suma de la fila redondeada, e igualmente con las columnas. Se hace referencia a este redondeo como un redondeo consistente.

Se trata de descubrir el esquema de redondeo resolviendo un problema de flujo factible con límites inferiores no negativos como capacidades en los arcos. Se construye la red asociada al problema y se halla el flujo máximo.

El procedimiento del problema se puede observar con el siguiente ejemplo. Supongamos que se tiene la matriz correspondiente al siguiente cuadro, en el que también se puede observar la suma de filas (en la cuarta columna) y la suma de las columnas (en la cuarta fila).

3,14	6,8	7,3	**17,24**
9,6	2,4	0,7	**12,7**
3,6	1,2	6,5	**11,3**
16,34	**10,4**	**14,5**	

Cuadro A.1: Matriz del problema con sumas de filas y columnas.

Se construye la red correspondiente a este problema del siguiente modo. Partiendo de la matriz anterior se asocia a cada arco la capacidad correspondiente a la parte entera y la parte entera +1.

A continuación, en dicha red los nodos de la forma i' se corresponden con cada fila de la matriz y los nodos de la forma j'' con cada columna de la matriz $\forall\, i', j'' = 1,2,3$.

Entonces el problema puede representarse del siguiente modo. Los arcos de entrada a la red $(1,i')\ \forall\ i' = 1,2,3$ se corresponden con las sumas de las filas, los arcos intermedios $(i', j'')\ \forall\ i', j'' = 1,2,3$ representan los cruces de las filas y columnas, y los arcos de salida $(j'',4)\ \forall\ j'' = 1,2,3$ se corresponden con las sumas de las columnas.

Es decir, gráficamente quedaría una red de la siguiente forma.

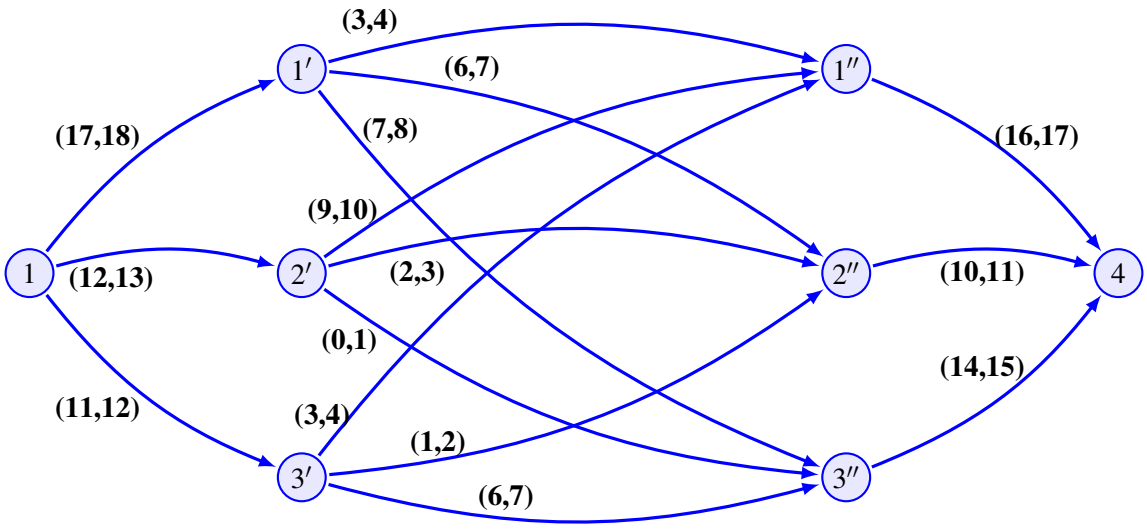

Figura A.5: Red del problema.

Resolviendo el problema como PMF se tiene que un posible redondeo es el siguiente:

3	7	7	**17**
10	2	1	**13**
3	1	7	**11**
16	**10**	**15**	

Cuadro A.2: Solución del problema.

Por tanto, se ha encontrado un redondeo consistente como requería el problema.

Apéndice B

Algoritmo de Ford y Fulkerson

El siguiente código es una modificiación del que puede consultarse en [9].

```
package FordFulkerson;

import java.util.LinkedList;
import java.util.Queue;

public class FordFulkerson {
    private int[] parent;
    private Queue<Integer> queue; //cola
    private int numberNodos;
    private boolean[] visited;//dar valor V o F
    private int[][] graph;
    private int source;
    private int sink;
    private int maxFlow;
    public FordFulkerson(int datos[][]) {
        this.numberNodos = datos[0][0];
        this.queue = new LinkedList<Integer>();
        parent = new int[numberNodos + 1];
        visited = new boolean[numberNodos + 1];
        graph = new int[numberNodos + 1][numberNodos + 1];
        for (int i = 1; i < datos.length; i++) {
            graph[datos[i][0] + 1][datos[i][1] + 1] = datos[i][2];
        }
        source = 1;
        sink = numberNodos;
    }
    public boolean bfs(int source, int goal, int graph[][]) { //alg de busqueda
        boolean pathFound = false;
        int destination, element;

        for (int vertex = 1; vertex <= numberNodos; vertex++) {
            parent[vertex] = -1;
            visited[vertex] = false;
        }
        queue.add(source); //añadir a la cola nodo origen
        parent[source] = -1;
        visited[source] = true;
        while (!queue.isEmpty()) {
            element = queue.remove();
            destination = 1;
            while (destination <= numberNodos) {
```

```
                if (graph[element][destination] > 0 && !visited[destination]) {
                    parent[destination] = element;
                    queue.add(destination);
                    visited[destination] = true;
                }
                destination++;
            }
        }
        if (visited[goal]) {
            pathFound = true;
        }
        return pathFound;
    }
    public int maxFlow() {
        int u, v;
        int maxFlow = 0;
        int pathFlow;
        int[][] residualGraph = new int[numberNodos + 1][numberNodos + 1];
        for (int sourceVertex = 1; sourceVertex <= numberNodos; sourceVertex++) {
            for (int destinationVertex = 1; destinationVertex <= numberNodos;
            destinationVertex++) {
                residualGraph[sourceVertex][destinationVertex] =
                graph[sourceVertex][destinationVertex];
            }
        }
        while (bfs(source, sink, residualGraph)) {
            pathFlow = Integer.MAX_VALUE;
            for (v = sink; v != source; v = parent[v]) {
                u = parent[v];
                pathFlow = Math.min(pathFlow, residualGraph[u][v]);
            }
            for (v = sink; v != source; v = parent[v]) {
                u = parent[v];
                residualGraph[u][v] -= pathFlow;
                residualGraph[v][u] += pathFlow;
            }
            maxFlow += pathFlow;
        }
        return maxFlow;
    }
}
```

Apéndice C

Algoritmo de preflujo

La obtención del máximo flujo para los algoritmos de preflujo se ha obtenido con una breve modificación del siguiente código [10].

```
package PushRelabel;
public class PushRelabel {
    int[][] capacidad;
    PushRelabel(int datos[][]) {
        //throw new UnsupportedOperationException("Not supported yet.");
        //Para cambiar el cuerpo de los métodos generados.
        init(datos[0][0]);
        for (int i = 0; i < datos[0][1]; i++) {
            addEdge(datos[i + 1][0], datos[i + 1][1], datos[i + 1][2]);
        }
    }
    public void init(int nodos) {
        capacidad = new int[nodos][nodos];
    }
    public void addEdge(int s, int t, int capacity) {
        capacidad[s][t] = capacity;
    }
    public int maxFlow(int s, int t) {
        int n = capacidad.length; //long capacidad
        int[] h = new int[n];
        h[s] = n - 1;
        int[] maxh = new int[n];
        int[][] f = new int[n][n];
        int[] e = new int[n];
        for (int i = 0; i < n; i++) {
            f[s][i] = capacidad[s][i];
            f[i][s] = -f[s][i];
            e[i] = capacidad[s][i];
        }
        for (int sz = 0;;) {
            if (sz == 0) {
                for (int i = 0; i < n; ++i) {
                    if (i != s && i != t && e[i] > 0) {
                        if (sz != 0 && h[i] > h[maxh[0]]) {
                            sz = 0;
                        }
                        maxh[sz++] = i;
                    }
                }
            }
            if (sz == 0) {
```

```
                break;
            }
            while (sz != 0) {
                int i = maxh[sz - 1];
                boolean pushed = false;
                for (int j = 0; j < n && e[i] != 0; ++j) {
                    if (h[i] == h[j] + 1 && capacidad[i][j] - f[i][j] > 0) {
                        int df = Math.min(capacidad[i][j] - f[i][j], e[i]);
                        f[i][j] += df;
                        f[j][i] -= df;
                        e[i] -= df;
                        e[j] += df;
                        if (e[i] == 0) {
                            --sz;
                        }
                        pushed = true;
                    }
                }
                if (!pushed) { //empuje flujo
                    h[i] = Integer.MAX_VALUE;
                    for (int j = 0; j < n; ++j) {
                        if (h[i] > h[j] + 1 && capacidad[i][j] - f[i][j] > 0) {
                            h[i] = h[j] + 1;
                        }
                    }
                    if (h[i] > h[maxh[0]]) {
                        sz = 0;
                        break;
                    }
                }
            }
        }
        int flow = 0;
        for (int i = 0; i < n; i++) {
            flow += f[s][i];
        }
        return flow;
    }
}
```